ROUTING-PROBLEM APPROACHES TO THE DESIGN OF A SYSTEM FOR URBAN WASTE COLLECTION

By

Ana Dolores López Sánchez

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR
IN THE DEPARTMENT OF ECONOMICS, QUANTITATIVE METHODS
AND ECONOMIC HISTORY
AT
PABLO DE OLAVIDE UNIVERSITY
AUGUST 2013

PABLO DE OLAVIDE UNIVERSITY
DEPARTMENT OF
ECONOMICS, QUANTITATIVE METHODS AND ECONOMIC HISTORY

The undersigned hereby certify that they have read and recommended to the Pablo de Olavide University for acceptance a thesis entitled "**Routing-Problem Approaches to the Design of a System for Urban Waste Collection** " by **Ana Dolores López Sánchez** in partial fulfillment of the degree of Doctor in the Department of Economics, Quantitative Methods and Economic History.

Dated: August 2013

Research Supervisor: _____
Prof. Dr. Alfredo García-Hernández Díaz

_____
Prof. Dr. Miguel Ángel Hinojosa Ramos

PABLO DE OLAVIDE UNIVERSITY

Date: **August 2013**

Author: **Ana Dolores López Sánchez**

Title: **Routing-Problem Approaches to the Design of a System for Urban Waste Collection**

Department: **Economics, Quantitative Methods and Economic History**

Signature of Author

To my family

INDEX

Acknowledgements

This thesis would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, I wish to thank Miguel Ángel Hinojosa Ramos and Alfredo García Hernández-Díaz , my supervisors, whom I consider to have been the most helpful and important people during this research. Without them, this work would never have come into existence. I thank them for their many suggestions, their unceasing patience and for their constant support from the beginning to the end of this work. It has been an honour for me to be able to work with them.

It would impossible forget my advisor abroad, Daniele Vigo, a good researcher and an even better person who, during my stay at Bologna University helped and taught me many new aspects of routing problems.

I am indebted to my many colleagues for their advice and supports. I thank Francisco Gortázar and Abraham Duarte for being always willing to assist me with the implementations.

I do not know how to show my gratitude to my family for their

love and patience. My parents have constantly supported my sister and I and encouraged us to always do our best. Thanks to my sister for giving me a hand when I needed it. Thanks Thibaud for being my confidant and my best friend, for your moral support at the most difficult moments and for sharing your life with me.

Finally, it also necessary include to all my friends. My dear friends, who together with my family, have suffered my absence and lack of time.

It is a pleasure to thank those who made this thesis possible.

Ana Dolores López Sánchez.

Seville, August 2013.

Preface

A routing problem arises when there is a set of demand points that have to be served by finding the "best" routes in order to provide the required service. Finding these "best" routes consists of optimizing one or several objective functions associated to these routes.

This Ph.D. thesis is motivated by a real-world routing problem. Specifically, we focus on the waste collection problem in the city of Seville and its resolution by using approximate algorithms. Waste collection is an interesting routing problem not only due to its high practical relevance, but also because it constitutes one of the most difficult operational problems faced by local authorities in any large city.

Seville has a population of $703,021$ inhabitants (according to official statistics of 2011) and is ranked as the fourth largest city of Spain. Urban waste is collected every single day of the year, i.e., seven days a week. The road network of Seville can be represented by a directed graph with $99,427$ points that correspond to street crossings and dead-end streets, $224,638$ fragments of streets delimited by two points and $1,642$ streets where there are containers placed. According to the local authority responsible for the waste collection in the city (February, 2012), 20 vehicles currently collect

the waste of $2,821$ containers placed at different locations all over the city. The annual waste generated amounts to approximately 23.2 litres per inhabitant.

This extremely tough problem can be described as follows. The waste generated in the city is the responsibility of the local authorities. Citizens deposit their refuse in containers along the streets and these containers must all be collected every day by a fleet of vehicles whose capacity cannot be exceeded. Each vehicle must travel along a route that is defined as a sequence of two or more trips due to the relatively small capacity of each vehicle, and the length of the working day must be satisfied. In the initial trip or open trip, vehicles leave the depot and start to collect refuse and once their capacity is full then go to the landfill. When vehicles have emptied the refuse into the landfill, they then begin as many closed trips as necessary without exceeding the working day. In the closed trips, vehicles start their trajectory from the landfill and return back to it when their capacity is full again. In closing, once vehicles have deposited the last load of refuse in the landfill they all return to the depot.

This Ph.D. thesis is organized as follows. In Chapter 1, a survey of routing problems and its variants is presented. We draw a distinction between vehicle (node) routing problems and arc routing problems. Furthermore, in order to equalize the two classes of problems, we describe well-known transformations and we propose a new and general transformation. This transformation can be seen as a generalization of the transformations that have been proposed in the literature so far.

In Chapter 2, background information on well-known solution methods is provided. In particular, a quick outline is drawn of the most popular, exact and approximate methods used to solve routing problems. Furthermore, a parallel is drawn between single-objective optimization problems, which find optimal or feasible solutions, and multi-objective optimization problems, which provide non-dominated or Pareto optimal solutions. We then describe performance metrics, which enables the quality of the solutions obtained to be measured. The main contribution of this chapter is that two versions of routing problems are solved by using new metaheuristics. Specifically, a new algorithm is designed to solve single-objective routing problems, in which the objective is to minimize the total routing cost, and four algorithms are designed to solve multi-objective routing problems, in which we do not only minimize the total routing cost but also balance the routes. In particular, the proposed algorithms combine two metaheuristics: Greedy Randomized Adaptive Search Procedure and Variable Neighborhood Descent.

In Chapter 3, the current waste collection problem in Seville is presented progressively. First, we focus on describing and modelling the waste collection in this city. The real-world problem cannot be solved by using exact methods, and hence the validity of the formulation is proved by using a number of small and medium-sized literature problems. The limitation imposed on the solution of these problems by using exact methods leads us to use approximate methods to solve the real-world problem. The proposed algorithms presented in Chapter 2 are therefore

adapted to solve the real-world problem. Again, both approaches are taken into account: the single-objective problem, in which the objective is to minimize the total distance travelled by all the vehicles; and the multi-objective problem, in which the objectives include the minimization of the total distance as well as a balance across the working day of the workers in each route.

This dissertation draws to a close with conclusions being reached and future research being proposed.

Chapter 1

Routing Problems

1.1 Introduction

A routing problem generally involves transportation that provides a service (this service could be *delivery* or *collection*) of goods or people between an initial point and a final point (these points may match up). The objective is to design a set of *routes* to service a set of customers with the minimum operational cost and the maximum customer satisfaction.

Routing problems arise in a number of practical contexts, such as mail delivery, routing of salespeople, waste collection, snow removal, street cleaning, school bus routing, airline scheduling, electric meter reading, and electrical lines and gas mains inspection. Vast amounts of money are spent each year by governments and private enterprise on these operations.

Operations researchers have, for a long time, studied the structure of these problems and proposed workable solutions, sometimes yielding sizeable savings.

The majority of routing problems use a *road network* that is generally described through a *graph*, whose *edges* and *arcs* represent the road sections and whose *vertices* correspond to the road junctions. For this reason, in the first section of this chapter, a short introduction to Graph Theory will be given.

A routing problem may be classified as a node routing problem, frequently known as a Vehicle Routing Problem (VRP) or as an Arc Routing Problem (ARP), depending on where the demand occurs on the underlying graph. In VRPs, which comprise most traditional routing applications, demands are located at customer sites, represented by nodes on the graph. However, in ARPs, the service activity involves the traversal or coverage of an arc because customers are located all over the arc. See Section 1.3 and Section 1.4.

For a given routing problem application, there is no clear-cut rule for choosing between a node routing or an arc routing model. An arc routing problem may be transformed to an equivalent node routing problem and vice versa. Resolution methods of routing problems differ substantially between node and arc routing. It can be shown that a VRP can be transformed into an ARP simply by replacing each node that requires service with an edge, see [62], making the two classes of problems equivalent. In a similar way, ARPs can be transformed into VRPs, although this is less trivial as will be shown later on. In Section 1.4.3, ARP transformations are described.

Routing problems are widely used to deal with real-life situations.

Those problems are often set up with the single objective of minimizing the cost of the solution, despite the fact the majority of the problems are of a multi-objective nature. For instance, more than one cost (distance, time, economic cost) may be associated with a solution obtained solving a single-objective routing problem. Furthermore, other aspects exist that could be considered, such as balancing of workloads, customer satisfaction, and management of the fleet. In this chapter, a final section will extend single-objective routing problems to multi-objective routing problems.

1.2 A Graph Theory Introduction

An *undirected graph* is a pair $G = (V, E)$ where V is a finite set of *vertices* (or *nodes*, or *points*) and $E \subseteq V \times V$ is a set of unordered pairs of not necessarily distinct vertices from V called *edges* (or *lines*). Hence, $V = \{1, 2, \ldots, n\}$ is the set of vertices and $E \subseteq \{(i, j) : i, j \in V\}$ is the set of edges. The usual way to create an undirected graph is by drawing a dot for each vertex and joining two of these dots with a line if the corresponding two vertices form an edge (see Figure 1.1).

Vertices joined by an edge are called *adjacent*. They are also called the *ends* of the edge. An edge is said to be *incident* to its ends. If all the vertices of G are pairwise adjacent, then G is *complete*.

A *loop* is an edge whose ends are the same vertex. An edge is *multiple* if there is another edge with the same ends; otherwise it is *simple*. An undirected graph is a *simple graph* if it has no multiple edges or loops,

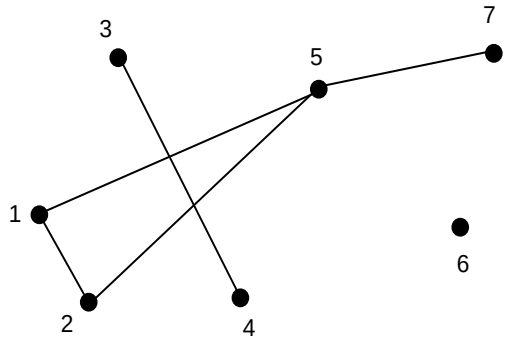


Figure 1.1: Undirected graph on $V = \{1, 2, 3, 4, 5, 6, 7\}$ with edge set $E = \{(1, 2), (1, 5), (2, 5), (3, 4), (5, 7)\}$

and it is a *multigraph* if it has multiple edges or loops.

A *directed graph* or *digraph* is a pair $G = (V, A)$ where V is a finite set of vertices and A consists of a set of ordered pairs of vertices, called *arcs*. The vertices i and j joined by an arc $(i, j) \in A$ are also said to be adjacent, with i being the *tail* of the arc and j being the *head* of the arc. The usual way to design a directed graph is by drawing a dot for each vertex and joining two of these dots with a line pointing to their orientation if the corresponding two vertices form an arc (see Figure 1.2).

A *mixed graph* has both edges and arcs. It is represented in the form $G = (V, E \cup A)$.

The *degree* of a vertex i in an undirected graph is the number of nodes adjacent to i. In the case of directed graphs, the *in-degree* of a vertex i is the number of arcs having i as head, and the *out-degree* of a vertex i is the number of arcs having i as tail.

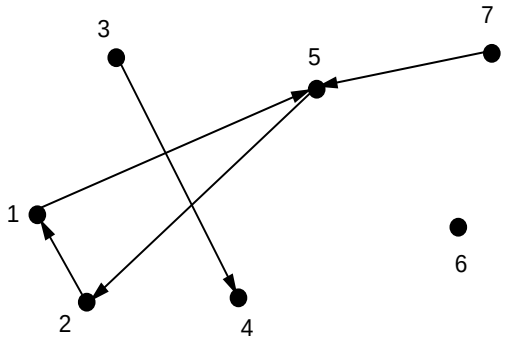


Figure 1.2: Directed graph on $V = \{1, 2, 3, 4, 5, 6, 7\}$ with edge set $A = \{(2, 1), (1, 5), (5, 2), (3, 4), (7, 5)\}$

1.2.1 Walks, Trails, Paths and Cycles

Given an undirected graph $G = (V, E)$, a directed graph $G = (V, A)$ or a mixed graph $G = (V, E \cup A)$, a *walk* from i to j, $W(i, j)$, is an alternating sequence of vertices and arcs and/or edges of G, $i_1, (i_1, i_2), i_2, (i_2, i_3), i_3, \ldots, (i_{m-1}, i_m), i_m$, such that $i_1 = i$ and $i_m = j$. The number of times that an arc or edge appears in a walk is termed its *multiciplicity*. If the walk ends in the same vertex that it starts with, i.e. $i = j$, then $W(i, j)$ is called a *closed walk*, otherwise it is an *open walk*.

If no arc or edge appears more than once in a walk, then it is called a *trail*. A closed trail is called a *tour* or a *circuit*.

Definition 1.2.1. *A Euler trail is a walk that traverses every edge of a graph exactly once. A graph is Eulerian if it admits a Euler circuit.*

If no vertex appears more than once in an open trail then it is called a *path*. A closed path (except the first and final vertex that coincide) and

with at least one arc or edge is called a *cycle* or a *simple circuit*.

Definition 1.2.2. *A Hamilton path is a walk that contains every vertex of a graph exactly once. A graph is Hamiltonian if it has a Hamilton cycle.*

A graph without any cycles is called a *forest*. A connected forest is called a *tree*. A forest is a graph whose connected components are trees.

A *weighted graph* is a graph in which each arc or edge, (i, j), has an associated non-negative real number, c_{ij}. If there is no arc or edge joining vertex i and j, then c_{ij} does not exist or it is infinity. Let C be the $n \times n$-matrix that represents the weights of each arc. Matrix C is called *the weight matrix*.

The *length* of a walk in a weighted graph is the sum of the weights of all the arcs or edges of the walk.

The *distance* between two vertices i and j in a weighted graph G, $d_G(i, j)$, is the length of the shortest path joining the vertices i and j.

From now on, only weighted graphs will be considered since most routing problems are defined on weighted graphs that represent a road network.

1.3 Vehicle Routing Problem

The earliest documented reference to **VRPs** was treated in 1856 by the Irish mathematician Sir William Rowan Hamilton and by the British mathematician Thomas Penyngton Kirkman and it was called the Travelling Salesman Problem (TSP). In the TSP, a number of cities have to be

visited by a salesman who must return to the same city that he started from, and each city has to be visited exactly once. When the graph G is completely directed, the problem is known as asymmetric TSP; when G is completely undirected, the problem is known as symmetric TSP. In solving the problem, one strives to construct the route so that the total distance travelled is minimized. A clear discussion of the early work of Hamilton and Kirkman can be found in [17]. The general form of the TSP was first studied by mathematicians, starting in the 1930s by Karl Menger in Vienna and Harvard. The problem was later promoted by Hassler Whitney and Merrill Flood at Princeton.

The multiple Travelling Salesman Problem is a generalization of the TSP, where more than one salesman is allowed to be used in the solution. The multiple TSP is an important problem in both theoretical and practical terms. First of all, it generalizes the TSP and can be studied to achieve a better understanding of the TSP from a theoretical point of view. On the other hand, by incorporating additional side constraints, such as capacity, distance and time-window restrictions, it could easily be extended to a variety of VRPs.

Theoretical research and practical applications in the field of VRP started in 1959 with the *Truck Dispatching Problem* posed by Dantzig and Ramser [34]: find the "*...optimum routing of a fleet of gasoline delivery trucks between a bulk terminal and a large number of service stations supplied by the terminal*". This problem is a generalization of the TSP. Using a method based on a linear programming formulation, their hand

calculations produced a near optimal solution with *four routes* to a problem with *twelve service stations*. The author proclaimed: "*No practical application of the method has been made as yet*". Since then the interest in VRP has been raised by a broad range of researchers and practitioners from different disciplines who are involved in this field today. There are major advances and new challenges thanks to technological innovation. Researchers and practitioners have developed faster, more accurate solution algorithms and better models that give them the ability to solve large-scale problems.

Indeed, the Truck Dispatching Problem is a Capacitated Vehicle Routing Problem (CVRP) because trucks are limited in their capacity. This problem has been studied in many variants. In fact, the majority of real-world problems are often much more complex than the CVRP that is also known as the *classical VRP*.

1.3.1 The Capacitated Vehicle Routing Problem

In the classical VRP, also known as the Capacitated Vehicle Routing Problem (CVRP), a fleet of identical vehicles is available in a depot to serve a set of customers. All the customers correspond to deliveries, and the demands are deterministic, known in advance, and cannot be split. Each customer is required to be served by exactly one vehicle. Thus the demand of each customer does not exceed the capacity of the vehicle. The goal is to find a set of routes in order to minimize the cost of the service.

A CVRP can be modelled by a graph where the vertex set is

$\{0, 1, \ldots, n\}$. Vertices $i = 1, \ldots, n$ correspond to the customers, whereas 0 corresponds to the depot.

The weight, c_{ij}, associated with each arc or edge (i, j) will be called *cost* of the arc or edge. The weight matrix, C, will be called the *cost matrix*. In some contexts, the cost can be interpreted as a *travel cost*, a *travel distance*, or a *travel time* spent travelling from vertex i to vertex j. If G is a directed graph, the cost matrix C is asymmetric, and the corresponding problem is called *asymmetric CVRP* (ACVRP). Otherwise, we have $c_{ij} = c_{ji}$ for each edge (i, j), and the problem is called *symmetric CVRP* (SCVRP).

Each customer i is associated with a known non-negative *demand*, q_i, to be delivered, whereas the depot 0 has a fictitious demand $q_0 = 0$. Given a customer set $S \subseteq V$, let $q(S) = \sum_{i \in S} q_i$ denote the total demand of the set S.

A set of K identical vehicles, each with capacity Q, is available at the depot. To ensure feasibility, we assume that $q_i \leq Q$ for each $i = 1, \ldots, n$. Each vehicle may perform at most one route, and we assume that K is no smaller than K_{min}, where K_{min} is the minimum number of vehicles needed to serve all the customers.

Given a set $S \subseteq V \setminus \{0\}$, we denote, by $r(S)$, the minimum number of vehicles needed to serve all customers in S. In the CVRP, note that $r(V \setminus \{0\}) = K_{min}$. It is worth mentioning that the value of $r(S)$ depends on the type of VRP under consideration; for instance, in the CVRP, it is valid to take the trivial *Bin Packing Problem* (BPP) lower bound

$\lceil q(S)/Q \rceil$, where for each real number x, $\lceil x \rceil$ is the ceiling function. In other variants of the VRP, $r(S)$ is not easy to determine *a priori* and the value may be determined by solving the BPP.

The CVRP consists of finding a collection of exactly K simple circuits or cycles (each corresponding to a vehicle route) of minimum cost, defined as the sum of the cost of the arcs belonging to the circuits, such that

1. each circuit begins and ends at the depot vertex;
2. each customer vertex is visited by exactly one circuit; and
3. the sum of the demand of the vertices visited by a circuit does not exceed the vehicle capacity, Q.

Several variants of the basic versions of the CVRP have been considered in the literature (see Section 1.3.3).

1.3.2 Basic Models for the CVRP

There are various formulations that can be used to model the CVRP. According to Toth and Vigo, see [128], three mathematical programming formulations are used as a base for the most recent exact solution approaches: vehicle flow formulations, commodity flow formulations, and set-partitioning formulations.

The models of the first family, known as *vehicle flow formulations*, use integer variables, associated with each arc or edge of the graph, which

count the number of times the arc or edge is traversed by a vehicle. These are the most frequently used models for the basic version of VRP. They are particularly suited for cases in which the cost of the solutions can be expressed as the sum of the costs associated with the arcs but they cannot be used to handle many practical issues.

The second family of models is based on the so-called *commodity flow formulation*. Here, additional variables are associated with the arcs or edges and represent the flow of the commodities along the paths travelled by vehicles.

The models of the third family are formulated as a *set-partitioning problem*. These models consider the set of all feasible circuits, whereby the binary coefficients are equal to 1 if and only if a vertex is visited on a circuit, i.e, each binary variable is associated with a different feasible circuit representing whether a route is used in the optimal solution. The main advantage of this type of model is that it is a very general model that can easily take several constraints into account. Moreover, the linear programming relaxation of this formulation is typically very tight.

As the number of formulations for each family is very large, we will only provide three representative examples: the two-index vehicle flow formulation, the two-commodity flow formulation, and the set-partitioning formulation.

An integer linear programming formulation for ACVRP, which could be adapted easily to SCVRP, is a *two-index vehicle flow formulation* originally proposed by Laporte *et al.* in 1985, see [88]. This model uses

binary variable x_{ij} that takes value 1 if arc $(i, j) \in A$ belongs to the optimal solution, and take value 0 otherwise.

$$\min \quad \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{1.3.1}$$

$$\text{s.t.} \quad \sum_{i \in V} x_{ij} = 1 \quad \forall j \in V \setminus \{0\}, \tag{1.3.2}$$

$$\sum_{j \in V} x_{ij} = 1 \quad \forall i \in V \setminus \{0\}, \tag{1.3.3}$$

$$\sum_{i \in V} x_{i0} = K, \tag{1.3.4}$$

$$\sum_{j \in V} x_{0j} = K, \tag{1.3.5}$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(S), \quad \forall S \subseteq V \setminus \{0\}, S \neq \emptyset, \tag{1.3.6}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in V. \tag{1.3.7}$$

The in-degree and out-degree constraints (1.3.2) and (1.3.3) mean that exactly one arc enters and leaves each vertex associated with a customer, respectively. Analogously, constraints (1.3.4) and (1.3.5) determine the degree requirements for the depot vertex.

The so-called *capacity-cut* constraint of (1.3.6) impose both the connectivity of the solution and the vehicle capacity requirements. The capacity-cut constraints remain valid also if $r(S)$ is replaced by the trivial BPP lower bound, see for instance, Cornuéjols and Harche [33].

An alternative formulation may be obtained by transforming the capacity-cut constraints (1.3.6), by means of the degree constraints (1.3.2)-(1.3.5), into the well-known *generalized subtour elimination* constraints:

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - r(S), \quad \forall S \subseteq V \setminus \{0\}, S \neq \emptyset, \tag{1.3.8}$$

which determine that at least $r(S)$ arcs leave each customer set S, see [86].

Both families of constraints, (1.3.6) and (1.3.8), have a cardinality growing exponentially with n.

A family of constraints, equivalent to (1.3.6) and (1.3.8), that has a polynomial cardinality, may be obtained by considering the subtour elimination constraints proposed for the TSP by Miller, Tucker and Zemlin in [100] and extending them to the ACVRP (see, for example, Christofides *et al.* [28], and Desrochers and Laporte [36]).

$$u_i - u_j + Q x_{ij} \leq Q - q_j, \quad \forall i, j \in V \setminus \{0\}, i \neq j, \tag{1.3.9}$$

such that $q_i + q_j \leq Q$,

$$q_i \leq u_i \leq Q, \quad \forall i, j \in V \setminus \{0\}, \tag{1.3.10}$$

where u_i for all $i \in V \setminus \{0\}$ are additional continuous variables representing the load of the vehicle after visiting customer i.

Two-index vehicle flow models have been used extensively to model the basic versions of the CVRP, but they are largely inadequate for variants of the VRP of a more complex nature. In fact, the two-index models can be used only when the cost of the solution can be expressed as the sum of the costs associated with the traversed arcs. In addition, it is not possible

to ascertain which vehicle traverses an arc used in the solution. Hence, the two-index models are not appropriate for the cases where the cost of a circuit depends on the overall vertex sequence or on the type of vehicle allocated to the route.

One possible way to overcome some of the drawbacks associated with two-index models is to explicitly indicate the vehicle that traverses an arc, so that more involved constraints may be imposed on the routes. In this way one obtains the so-called *three-index vehicle flow formulation* given by Golden *et al.* in 1977, see [65]. This model uses binary variable x_{ijk}, which takes value 1 if vehicle k travels directly from customer i to customer j, and 0 otherwise; and variable y_{ik}, which takes value 1 if customer i is served by vehicle k in the optimal solution, and takes value 0 otherwise. Three-index vehicle flow models have been extensively used to model more constrained versions of the CVRP but the main drawback is represented by the increased number of variables. On the other hand, they generalize the two-index models. Close attention to the three-index vehicle flow formulation can be found in Chapter 1 of the book by Toth and Vigo [128].

Another family of models is called *commodity flow formulation*. Specifically, the two-commodity flow formulation of the CVRP was proposed by Baldacci *et al.* in 2004, see [6], and is based on the two-commodity flow formulation introduced by Finke *et al.* in 1984, see [51], for the TSP. These formulations, in addition to the variables used by the two-index vehicle flow formulations, require a new set of (continuous) variables,

associated with the arcs, which represent the amounts of demand that flow along them. Since commodity flow formulation explicitly introduces arc orientation even in symmetric problems, we describe the model for ACVRP.

The formulation requires the extended graph $G' = (V', A')$ obtained from G by adding vertex $n+1$, which is a copy of the depot node. Routes are now paths from vertex 0 to the vertex $n+1$. Two non-negative *flow variables*, f_{ij} and f_{ji}, are associated with each edge $(i,j) \in A'$. If a vehicle travels from i to j, then f_{ij} and f_{ji} give the vehicle load and the vehicle residual capacity, respectively, along the arc, i.e., equation $f_{ji} = Q - f_{ij}$. The roles are reversed if the vehicle travels from j to i. Therefore, the equation $f_{ij} + f_{ji} = Q$ holds for each $(i,j) \in A'$.

As in the two-index vehicle flow formulation for each $(i,j) \in A'$, let x_{ij} be equal to 1 if the arc is in the solution and be equal to 0 otherwise. An integer formulation for ACVRP is therefore as follows:

$$\min \quad \sum_{(i,j)\in A'} c_{ij}x_{ij} \tag{1.3.11}$$

$$\text{s.t.} \quad \sum_{j\in V'}(f_{ji} - f_{ij}) = 2q_i \quad \forall i \in V' \setminus \{0, n+1\}, \tag{1.3.12}$$

$$\sum_{j\in V'\setminus\{0,n+1\}} f_{0j} = q(V \setminus \{0, n+1\}), \tag{1.3.13}$$

$$\sum_{j\in V'\setminus\{0,n+1\}} f_{j0} = KQ - q(V \setminus \{0, n+1\}), \tag{1.3.14}$$

$$\sum_{j\in V'\setminus\{0,n+1\}} f_{n+1j} = KQ, \tag{1.3.15}$$

$$f_{ij} + f_{ji} = Q, \quad \forall (i,j) \in A', \tag{1.3.16}$$

$$\sum_{j \in V'} (x_{ij} - x_{ji}) = 2 \quad \forall i \in V' \setminus \{0, n+1\}, \tag{1.3.17}$$

$$f_{ij} \geq 0 \quad \forall (i,j) \in A', \tag{1.3.18}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A'. \tag{1.3.19}$$

Flow conservation constraints (1.3.12) means that the difference between the sum of the commodity flow variables associated with arc entering and leaving each vertex i is equal to twice the demand of i. Constraints (1.3.13)-(1.3.15) impose the correct values for the commodity flow variables incident into the depot vertices. Finally, constraints (1.3.16) and (1.3.17) determine the relation between vehicle flow and commodity flow variables and the vertex degree, respectively.

To close this section, VRP will be formulated as a set-partitioning problem. The *set-partitioning formulation* of the VRP was originally proposed by Balinsky and Quandt, see [8], as being necessary to use binary variables, each associated with a different feasible circuits of G. Therefore, let $\mathcal{H} = \{H_1, \ldots, H_q\}$ denote the collection of all the circuit of G, each corresponding to a feasible route, with $q = |\mathcal{H}|$. Each H_j has an associated cost c_j. In addition, let a_{ij} be a binary coefficient that takes value 1 if vertex i is visited by route H_j, and takes value 0 otherwise. The binary variable x_j, $j = 1, \ldots, q$ is equal to 1 if and only if circuit H_j is selected in the optimal solution. The model is

$$\min \quad \sum_{j=1}^{q} c_j x_j \tag{1.3.20}$$

$$\text{s.t.} \quad \sum_{j=1}^{q} a_{ij} x_j = 1 \quad \forall i \in V \setminus \{0\}, \tag{1.3.21}$$

$$\sum_{j=1}^{q} x_j = K, \tag{1.3.22}$$

$$x_j \in \{0,1\} \quad \forall j = 1, \ldots, q. \tag{1.3.23}$$

Constraints (1.3.21) mean that each customer i is covered by exactly one of the selected circuit, and (1.3.22) requires that K circuits are selected. This is a very general model that may easily take into account other constraints since route feasibility is implicitly considered in the definition of the set $\mathcal{H}$. Moreover, the linear programming relaxation of this formulation is very tight.

One of the main drawbacks of this model is its huge number of variables. The explicit generation of all the feasible circuits is thus normally impractical, and one has to resort to solving the linear programming relaxation of the model.

1.3.3 VRP Variants

In this section we refer to the problem variant as follows:

• **Distance Constrained Vehicle Routing Problem (DVRP).** This problem is an extension of the CVRP, where each route has a maximum length (or time) constraint. In particular, a non-negative *length*, t_{ij}, is associated with each arc or edge (i, j), and the total length of the arcs of each route cannot exceed the maximum route length, T. Generally, the cost and the length matrices coincide, i.e., $c_{ij} = t_{ij}$ for all arcs or edges.

Hence, the objective of the problem is to minimize the total length of the routes or their duration, when the service time is included in the travel time of the arcs or edges. The case in which both the vehicle capacity and the maximum distance constraints are present is called a Distance Constrained Vehicle Routing Problem (DVRP). Practical applications of DVRP can be found in Assad [2] and Laporte *et al.* [87].

• **Open Vehicle Routing Problem (OVRP).** The OVRP is a special case of the CVRP where vehicles do not need to return to the depot after visiting the last customer of a given route. Most authors minimize the number of routes as a first objective and then minimize the total cost. Several others authors just minimize the total cost without taking the number of vehicles into account.

Applications of the OVRP may arise when a company chooses to hire a vehicle fleet to give a service to their customers, then, due to logistic reasons, these vehicles are not forced to return to the company's depot. A further possible application is the planning of train or bus services.

The earliest publication about the OVRP can be traced back to the article by Schrage in 1981, see [122], which classifies the features encountered in practical routing problems. Schrage was the first to distinguish between closed trips travelled by private vehicles, and open trips assigned to common carrier vehicles. However, the first solution approach for the OVRP is thanks to Bodin *et al.* in 1983, see [19]. Their paper deals with a real-life application, an express airmail distribution problem.

• **Heterogeneous Fleet Vehicle Routing Problem (HFVRP).**

This problem is a generalization of the classical VRP because it considers vehicles with different capacities, instead of simply a homogeneous fleet. There are several HFVRP variants in the literature, and are basically related to the fleet limitation (limited or unlimited) and the costs considered (dependent and/or fixed). The HFVRP with unlimited fleet, also known as the Fleet Size and Mix (FSM), was proposed by Golden *et al.* in 1984, see [63], and consists of determining the best fleet composition and its optimal routing scheme. Another HFVRP version, called Heterogeneous VRP (HVRP), was proposed by Taillard in 1999, see [126], and consists of optimizing the available fixed fleet.

This situation can often be found in practice, and hence the HFVRP may be a suitable model for dealing with real-world applications. The fleet of vehicles in a company is rarely homogeneous. Generally, either an acquired fleet is already heterogeneous or they become heterogeneous over time due to the incorporation of vehicles with different features into the original fleet.

• **Multi-depot Vehicle Routing Problem (MDVRP).** Here each vehicle is located in one of several depots from which it must start and end its tour. The most common variation is where each vehicle must return to the depot from which it originated, but one could also consider the case where each vehicle just has to return to any depot independent of its starting location.

Applications of the MDVRP may arise when a company has multiple warehouses, each of which own vehicle fleet that together are capable of

meeting customers' demands.

Tillman, see [127], was the first to address the MDVRP in late 1960's.

• **Vehicle Routing Problem with Time Windows (VRPTW).** This is another extension of the CVRP, in which capacity constraints are imposed, but there is also a specific time period to serve each customer, i.e., a time interval or a time window $[a_i, b_i]$ and a service time s_i is associated with node $i \in V$. In the hard time window variant, the node must be served within that interval (although the vehicle can wait, if it arrives before the lower bound). In the soft time window variant, the node can be served outside its time interval, but a penalty is incurred in the objective if the vehicle arrives after the upper bound. A waiting time is introduced in the route schedule when the vehicle arrives before the lower bound. The VRP with time Deadlines (VRPD) is a special case of the VRPTW where only an upper time bound is associated with each customer.

This situation appears in grocery distribution or other businesses that are open only during working hours. The vehicles can therefore give their services only during this interval of time.

Early work on the VRPTW was case-study oriented proposed by Pullen and Webb in 1967, see [115], and one year later by Knight and Hofer, see [84].

• **Periodic Vehicle Routing Problem (PVRP).** The PVRP is a generalization of the CVRP in which vehicle routes must be constructed

over multiple days. During each day within a planning period, a fleet of capacitated vehicles travel along routes that begin and end at a single depot.

Examples of companies offering periodic services to the customers include those for grocery distribution and waste collection.

The PVRP was first proposed within the waste collection context by Beltrami and Bodin in 1974, see [16], in which vehicle routes are constructed over a horizon of t periods.

• **Split Delivery Vehicle Routing Problem (SDVRP).** The SDVRP involves a situation where a customer needs to be served more than once, contrary to what is usually assumed in the classical VRP. Therefore, the demand of each customer may be greater than the vehicle capacity. The problem consists of finding a set of vehicle routes that serve all the customers such that both the sum of the quantities delivered in each tour does not exceed the capacity of a vehicle, and the total distance travelled is minimized.

The SDVRP was introduced in the literature by Dror and Trudeau (see [41] and [42]) who motivated the study of the SDVRP by showing that there can be savings generated by allowing split deliveries.

Applications one can find in the literature include the problem of managing a fleet of trucks for distributing feed in a large livestock ranch in Arizona; routing of helicopters for weekly crew exchanges at natural gas platforms in the North Sea; distribution of bundles of newspapers in Seoul; and containerized sanitation pickup at commercial office buildings.

• **Vehicle Routing Problem with Backhauls (VRPB).** This constitutes a further variant of the CVRP in which the set of nodes is partitioned into two subsets: linehaul nodes for which the demand is delivered from the depot, and backhaul nodes for which the demand is picked up and brought back to the depot. It has been quickly recognized that substantial cost savings can be achieved by allowing empty vehicles to pick up inbound products when they return to the depot. In classical VRPB, there is a strict precedence relationship between the linehaul and backhaul nodes, that is, all linehauls must be visited before backhauls. Without this constraint, goods could be picked up while other goods would not have yet been delivered, thus potentially leading to a rearrangement of the goods inside the vehicle. Finally, in the mixed VRPB, linehaul and backhaul nodes can be freely mixed as long as the capacity constraint is satisfied.

A real-world application is the grocery industry, where supermarkets and shops are linehaul nodes and grocery suppliers are the backhauls nodes.

• **Vehicle Routing Problem with Simultaneous Pickup and Delivery (VRPSPD).** Here the vehicle has to pick something up and deliver it to the customer, simultaneously or not. Hence, each customer request corresponds to a pair of nodes, node $i+$ where the demand must be picked up and node $i-$ where the demand must be delivered. Both nodes must be in the same vehicle route and node $i+$ must be visited before $i-$.

A number of applications of the VRPSPD can be found in the beverage industry, where filled bottles are delivered while the empty ones are collected; in grocery stores, where pallets or containers are collected for re-use in merchandise transportation, etc.

The VRPSPD was first proposed by Min in 1989, see [99]. The author presented a real-life problem concerning the distribution and collection of books of a public library.

• **Vehicle Routing Problem with Multiple Use of vehicles.** In VRP, it is implicitly assumed that each vehicle serves a single route. In some cases, however, it might be possible or even necessary to assign the vehicle to several routes. This situation happens, for example, when the capacity of the vehicle is relatively small and the working time of each vehicle must be satisfied. This variant is also known as Vehicle Routing Problem with Multiple Trips.

An example could correspond to an eight-hour working day. In several contexts, once the vehicle routes have been designed, it may be possible to assign several of them to the same vehicle.

The VRP with Multiple Use of vehicles was explicitly addressed in a work presented by Fleischmann in 1990, see [52].

• **Dynamic Vehicle Routing Problem**, which certain data about the problem is not known beforehand. That is, new information is revealed on-line, as the routes are executed by the vehicles. The new information often corresponds to the occurrence of a new node (customer) that must be included into the current routes. This data can also be in the form of

new information about the travel time of a vehicle, or the current customer status. Psaraftis, see [114], examines the main issues in Dynamic VRPs.

Applications include pickup and delivery of overnight mail, the distribution of heating oil or liquid gas to private households, residential utility repair services, such as cable and telephone, and appliance repair. Other potential applications involve taxi cab services and emergency services.

• **Stochastic Vehicle Routing Problem (SVRP).** This constitutes another difficult variant of VRP that requires demands, travelling times, or customers to be a random variable. For example, in VRPs with stochastic demand, a probability distribution is specified for the demand of each customer and it is usually assumed that demands are independent; however this may not always be realistic. Another example of a variant in VRPs with stochastic customers is that each customer has a given probability of requiring a visit. See [56] that summarizes the scientific literature on stochastic vehicle routing problems.

Further variants of the VRP can be found in the literature combining the above, and other more elaborate obtained by variants of the VRP.

Before finishing this section, various surveys on the VRP must be named, such as the book by Toth and Vigo, see [128], or the book by Golden *et al.*, see [61]. Specific surveys of rich VRPs may be found in Bräysy *et al.*, see [24].

1.4 Arc Routing Problem

The earliest documented reference to ARPs began in 1735: the famous Königsberg bridge problem. The problem can be represented by an undirected graph in which one vertex is used for each of the two shores of the river and for the two islands, and each edge corresponds to a bridge (see Figure 1.3). The question was to determine whether there exists a closed walk traversing *exactly once* each of seven bridges on the Pregel river in Königsberg, now called Kaliningrad. The problem was solved by the Swiss mathematician Leonhard Euler who showed there was none in this particular case, see [47]. In fact, Euler illustrated that for a closed walk in an undirected graph to exist, all vertices must have an even degree. Euler was concerned almost exclusively with the existence of a closed walk, he was not apparently concerned with the problem of actually determining a closed walk in an Eulerian graph. This problem was addressed more than a century later by Hierholzer in 1973 (see [74]) and the sufficiency of this condition was proved.

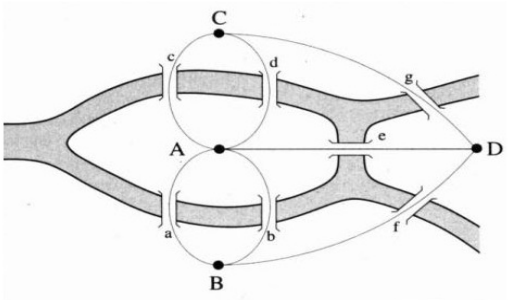


Figure 1.3: The seven bridges od Königsberg

A well-known problem, closely related to the problem presented by Euler, is the so-called Chinese Postman Problem (CPP) posed by Meigu Guan in 1962, a mathematician at the Shangtun Normal College who spent some time as a post office worker during the Chinese cultural revolution. In contrast to the Königsberg bridge problem, which is only concerned with the existence and determination of a closed walk traversing each edge exactly once, here the question is to deal with situations where such a solution does not exist. Guan addressed the question of minimizing the length of a walk passing through each edge of an undirected graph *at least once*. The problem is stated in a simple way by Guan [67]: "*A mailman has to cover his assigned segment before returning to the post office. The problem is to find the shortest walking distance for the mailman.*"

Another CPP variant is the Windy Postman Problem, which was first introduced by Minieka in 1979, see [101]. The problem is defined on an undirected graph where two different costs are associated with each edge, as with and against the wind. The main difference is that, in this problem, the cost of traversing an edge depends on the direction travelled. As the CPP, the Windy Postman Problem consists of determining a least-cost traversal of all edges of the graph.

Another CPP extension is the hierarchical CPP first introduced by Dror *et al.* in 1987, see [40], where a precedence relation is defined on arcs or edges, and the order in which arcs or edges are serviced must respect this relation. The hierarchical CPP can be defined on a directed, undirected

or mixed graph and consists of determining a least-cost traversal of the graph starting at a point s and ending at a point t and servicing all the arcs or edges, which obeys the aforementioned precedence relation.

Another major CPP is the Min-Max k-CPP, as suggested by Frederickson *et al.* in 1978, see [54]. The Min-Max k-CPP is defined on an undirected graph and consists of finding k tours, starting and ending in a special node called the depot node, such that every edge is covered by at least one tour and the length of the longest tour is minimized. It should be noted that, for this problem, the objective is to minimize the *makespan*, i.e., the length of the longest tour, whereas most other problems with multiple postmen seek to minimize the total distance travelled.

When it is required to traverse only a subset of arcs or edges, then the problem becomes the Rural Postman Problem (RPP). The problem was introduced by Orloff (1974), see [107], and is formally stated as follows: *Given an undirected graph, find a minimum cost tour, which passes through every edge in a subset $E_R \subseteq E$ at least once.* The subset E_R is called required edges.

A constrained version of the RPP is called Capacitated Arc Routing Problem (CARP) introduced by Golden and Wong in 1981, see [62], in which a set of vehicles are limited in their capacity. The CARP is one of the most studied ARPs due to its practical applications. Note that the Capacitated CPP is a variation of the CARP where every edge in the graph has a strictly positive demand; this was first proposed by Christofides in 1973, see [27].

In order to complete this section, certain ARP surveys are worth mentioning, such as Assad and Golden [3], Dror [39], and Eiselt *et al.* [44] and [45].

The variants described in VRPs can also be extended to ARPs.

1.4.1 The Capacitated Arc Routing Problem

The Capacitated Arc Routing Problem (CARP) is very similar to the CVRP. In the CARP, a fleet of identical vehicles is available in a depot to serve a set of arcs or/and edges of a graph instead of serving a set of vertices as in the CVRP. It is worth mentioning that in the CVRP, the customers are situated at the vertices, and in the CARP, the customers are situated along the arc or edge. Each arc or edge with demand must be served by exactly one vehicle and the goal is to find a set of routes which incur the least cost. Note that in the CARP, a vehicle could traverse an arc or edge without servicing it, but each arc or edge must be serviced by exactly one vehicle, while in the CVRP, all vertices must be service exactly once by one vehicle.

Again, a CARP can be modelled by a connected graph where $\{0, 1, \ldots, n\}$ is the vertex set with a distinguished depot vertex labelled 0. The weight, c_{ij}, is associated with each arc or edge (i, j) and will be called *traversal cost* or *deadheading cost*. There is a subset of required arcs $A_R \subseteq A$ or of required edges $E_R \subseteq E$ with demands q_{ij}.

A set of K identical vehicles, each with capacity Q, is available at the depot and we assume that $q_{ij} \leq Q$ for each $(i, j) \in A_R$ or for each

$(i, j) \in E_R$ to ensure feasibility. Each vehicle may perform at most one route, and we assume that K is no smaller than K_{min}. The value of K_{min} was first determined by Golden and Wong (see [62]) and is computed as $\left\lceil \sum_{(i,j)\in A_R \cup E_R} q_{ij}/Q \right\rceil$, but any of the known BPP lower bounds could be used.

The CARP consists of finding a collection of exactly K closed walks (each corresponding to a vehicle route) with minimum cost, defined as the sum of the cost of the arcs or edges belonging to the closed walks, such that:

1. each walk begins and ends at the depot vertex;
2. each arc or edge with positive demand is traversed exactly once; and
3. the sum of the demand of the arcs or edges visited in each walk does not exceed the vehicle capacity, Q.

1.4.2 Basic Models for the CARP

There are a large number of competing formulations for the CARP in the literature. Since the first mathematical formulation of the CARP by Golden and Wong in 1981, [62], several formulations have been proposed for the problem, ranging from sparse formulations, to dense formulations, and to supersparse formulations, see Dror [39].

In real-life CARP instances, it is common for the number of vehicles to be small and moreover they are frequently defined on road networks, with the result that G is very sparse (most vertices are of a degree smaller

than 5). Under such circumstances it is natural to use the approach known as *sparse formulations*, in which, for each vehicle and each edge two binary variables are defined, whether the edge is traversal but giving service, or the edge is traversal without giving service.

When the number of vehicles is large, or when $|E_R|$ is small relative to $|E|$, then the graph G can be broken down as follows: a complete graph G' is constructed with two vertices for each required edge, representing the two endpoints, together with an extra vertex representing the depot. An edge from one vertex to another in the expanded graph represents the shortest path between the corresponding pair of vertices in G. This leads to what is called *dense formulations*, with $2|E_R|^2$ variables, and represent whether a vehicle traverses between two vertices for each required edge.

A third approach called *supersparse formulations* in which each variable represents the number of times a particular edge is traversed without being serviced. Such formulation is highly economical in the sense that uses only aggregated variables, which are elegant and easy to understand. However, it comes at a price: in a feasible solution, such formulation gives no indication of which vehicle traverses which edge.

Still other approaches to the CARP have been proposed. The first integer linear programming formulation for the CARP was proposed by Golden and Wong (1981) and is based on directed variables even though the problem formulated is undirected or mixed. As the number of formulations is very large, we will describe this one in detail.

In the directed CARP formulation given by Golden and Wong,

binary variables y_{ijk} are equal to 1 if and only if arc (i, j) is *traversed* from i to j by vehicle k, and binary variables x_{ijk} are equal to 1 if and only if (i, j) is *serviced* by vehicle k while travelling from i to j. This problem can therefore be formulated as follows:

$$\min \quad \sum_{k\in K}\sum_{(i,j)\in A} c_{ij}y_{ijk} \tag{1.4.1}$$

$$\text{s.t.} \quad \sum_{j:(i,j)\in A} y_{ijk} = \sum_{j:(j,i)\in A} y_{jik}, \quad \forall i \in V, \forall k \in K, \tag{1.4.2}$$

$$\sum_{k\in K} x_{ijk} = 1, \quad \forall (i,j) \in A_R, \tag{1.4.3}$$

$$y_{ijk} \geq x_{ijk}, \quad \forall (i,j) \in A, \forall k \in K, \tag{1.4.4}$$

$$\sum_{(i,j)\in A_R} q_{ij}x_{ijk} \leq Q, \quad \forall k \in K, \tag{1.4.5}$$

$$\left.\begin{array}{l} \sum_{i\in S}\sum_{j\in S} y_{ijk} - n^2 u_k^S \leq |S| - 1, \\ \sum_{i\in S}\sum_{j\notin S} y_{ijk} + w_k^S \geq 1, \\ u_k^S + w_k^S \leq 1, \\ u_k^S, w_k^S \in \{0,1\}, \end{array}\right\} \; S \subseteq V \setminus \{0\}, \; S \neq \emptyset, \forall k \in K, \tag{1.4.6}$$

$$y_{ijk} \in \{0,1\}, \quad \forall (i,j) \in A_R, \forall k \in K, \tag{1.4.7}$$

$$x_{ijk} \in \{0,1\}, \quad \forall (i,j) \in A, \forall k \in K. \tag{1.4.8}$$

The objective function represents the total distance travelled by all the vehicles. Note that it is unnecessary to add the service cost because it will be fixed cost. In this formulation, constraints (1.4.2) are flow conservation equations for each vehicle. Constraints (1.4.3) ensure that service arcs correspond to those with demand. Constraints (1.4.4) state that an arc is serviced by a vehicle only if it is traversed by the same vehicle. Constraints (1.4.5) guarantee that the capacity of a vehicle

is never exceeded. Constraints (1.4.6) ensure that the solution does not contain any illegal subtour. To see how these constraints operate, observe that for a given k and S, only one of the two binary variables u_k^S or w_k^S can take the value 1; for more detail, see [12]. Of course, the exponential number of constraints makes their direct use impractical.

It is worth mentioning that the previous formulation can be adapted easily to UCARP and MCARP by replacing (1.4.3) with $\sum_{k\in K}(x_{ijk} + x_{jik}) = 1$ for all $(i, j) \in E_R$, or by including both constraints for the MCARP.

1.4.3 ARP Transformations

As previously mentioned, it is possible to transform a node routing problem into an arc routing problem simply by replacing each node with one edge and assigning the demand of the original node to the edge. The two nodes of such an edge are adjacent to all the other nodes that the original node has been adjacent to, see [62] for details. However, this transformation of a routing problem is almost never executed in practice for obvious computational reasons. On the other hand, transformation in the opposite direction is even more difficult.

In this section we will examine a number of graph transformations for the CARP which accept a routing problem statement expressed in terms of arc or edge traversals and converts it to a node routing problem on a related graph. The node routing solution on the modified graph is equivalent to the arc or edge traversal solution on the original graph.

Pearn, Assad and Golden's Transformation

Let $G = (V, E)$ be an undirected graph. Pearn, Assad and Golden (see [111]) proposed a transformation of the UCARP to the SCVRP that replaces each required edge with three nodes. Each edge $(i, j) \in E_R$ is associated to nodes s_{ij} and s_{ji}, called side nodes, and to m_{ij}, called the middle node. A CVRP instance is defined on the complete undirected graph $\hat{G} = (\hat{V}, \hat{E})$, where $\hat{V} = \bigcup_{(i,j)\in E_R}\{s_{ij}, s_{ji}, m_{ji}\}\bigcup\{0\}$. Note that all nodes are discarded except the depot.

If $d_G : V \times V \longrightarrow \mathbb{R}$ denotes the distance function that assigns to every pair of vertices in V, $d_G(i, j)$ is the length of the shortest path from vertex i to vertex j in the graph G. The distance function, $d_{\hat{G}} : \hat{V} \times \hat{V} \longrightarrow \mathbb{R}$, is defined as follows:

$$d_{\hat{G}}(0, s_{ij}) = \frac{1}{4}c_{ij} + d_G(0, i)$$

$$d_{\hat{G}}(s_{ij}, s_{kl}) = \begin{cases} \frac{1}{4}(c_{ij} + c_{kl}) + d_G(i, k) & \text{if } (i, j) \neq (k, l) \\ 0 & \text{if } (i, j) = (k, l) \end{cases}$$

$$d_{\hat{G}}(m_{ij}, v) = \begin{cases} \frac{1}{4}c_{ij} & \text{if } v = s_{ij} \text{ or } v = s_{ji} \\ \infty & \text{otherwise} \end{cases}$$

The new demands are $q(s_{ij}) = q(m_{ij}) = q(s_{ji}) = \frac{1}{3}q_{ij}$. The allocation of the edge demand q_{ij} to these nodes can be made arbitrarily, one only needs to ensure that $q(s_{ij}) + q(s_{ji}) + q(m_{ij}) = q_{ij}$. As usual, the depot node has no demand associated with it.

Since, in the classical VRP, each node can only be visited by a single vehicle, this transformation guarantees that either the nodes s_{ij},

m_{ij}, s_{ji} or the nodes s_{ji}, m_{ij}, s_{ij} are visited in sequence on the same vehicle route for the VRP solution. This transformation prevents the generation of both solutions with infinite distances and those which visit either of the side nodes twice. To convert a single edge into three nodes adds only two edge segments: one needs to add (1/4)th of the original edge distance when entering side node s_{ij}, (1/4)th of the original edge distance when leaving side node s_{ji}, and half of the edge distance when going from s_{ij} to s_{ji} through m_{ij}, or vice versa. Note that nodes s_{ij}, m_{ij}, s_{ji} are also visited in the same order as the edge (i, j) is traversed in the CARP solution.

For an illustration of the transformation by Pearn, Assad and Golden, see Figure 1.4. The instance contains 4 nodes and 5 edges, where $V = \{0, 1, 2, 3\}$ is the set of nodes, $E = \{(0, 1), (0, 2), (1, 2), (1, 3), (2, 3)\}$ is the edge set, and $E_R = \{(0, 2), (1, 2), (1, 3), (2, 3)\}$ is the set of required edges. Table 1.1 provides the distances between the pairs of nodes for the example depicted in Figure 1.4.

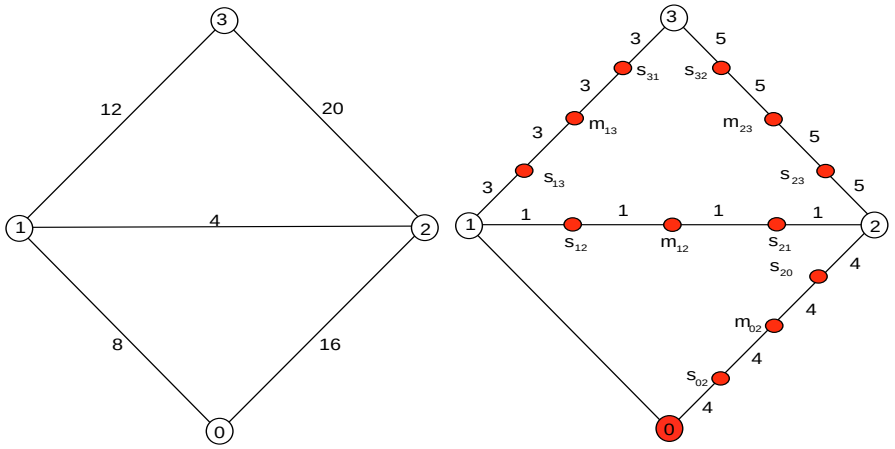


Figure 1.4: Pearn, Assad and Golden's Undirected Graph

	0	s_{02}	m_{02}	s_{20}	s_{12}	m_{12}	s_{21}	s_{13}	m_{13}	s_{31}	s_{23}	m_{23}	s_{32}
0	-	4	∞	16	9	∞	13	11	∞	23	17	∞	25
s_{02}	4	-	4	20	13	∞	17	15	∞	27	21	∞	29
m_{02}	∞	4	-	4	∞	∞	∞	∞	∞	∞	∞	∞	∞
s_{20}	16	20	4	-	9	∞	5	11	∞	23	9	∞	25
s_{12}	9	13	∞	9	-	1	6	4	∞	16	10	∞	18
m_{12}	∞	∞	∞	∞	1	-	1	∞	∞	∞	∞	∞	∞
s_{21}	13	17	∞	5	6	1	-	8	∞	20	6	∞	22
s_{13}	11	15	∞	11	4	∞	8	-	3	18	12	∞	20
m_{13}	∞	∞	∞	∞	∞	∞	∞	3	-	3	∞	∞	∞
s_{31}	23	27	∞	23	16	∞	20	18	3	-	24	∞	8
s_{23}	17	21	∞	0	10	∞	6	12	∞	24	-	5	26
m_{23}	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	5	-	5
s_{32}	25	29	∞	25	18	∞	22	20	∞	8	26	5	-

Table 1.1: Distance matrix according to Pearn, Assad and Golden's Transformation for an undirected graph

Pearn, Assad and Golden's transformation establishes the equivalence of the original arc routing problem and its node routing counterpart in which an UCARP with $|E_R|$ required edges results in a SCVRP with $3|E_R| + 1$ nodes.

When the number of required edges $|E_R|$ is small, relative to the number of edges $|E|$, then this transformation could prove itself to be useful because the node triplets are introduced only for those edges that have positive demand. Thus, in the case of the Capacitated Chinese Postman Problem transformation, the resulting number of nodes is roughly three times the number of edges in the original problem. This increase reflects the greater inherent difficulty of solving arc routing problems as compared to their node routing counterparts.

The transformation introduced by Pearn, Assad and Golden may be viewed as a means for linking the two classes of node and arc routing problems. As Eiselt *et al.* hold in 1995, see [45], the interest of this transformation is mostly formal and its algorithmic value has yet to be demonstrated.

Longo, Aragão and Uchoa's Transformation

Let $G = (V, E)$ be an undirected graph. Longo, Aragão and Uchoa (see [93]) proposed a transformation of the UCARP to the SCVRP that replaces each required edge with two nodes. They avoid the middle nodes proposed by Pearn, Assad and Golden, and hence each edge $(i, j) \in E_R$ is only associated to two side nodes s_{ij} and s_{ji}. The resulting CVRP

instance is defined on the complete undirected graph $\hat{G} = (\hat{V}, \hat{E})$, where $\hat{V} = \bigcup_{(i,j)\in E_R}\{s_{ij}, s_{ji}\}\bigcup\{0\}$.

The distance function is defined as follows:

$$d_{\hat{G}}(0, s_{ij}) = d_G(0, i)$$

$$d_{\hat{G}}(s_{ij}, s_{kl}) = \begin{cases} 0 & \text{if } (i,j) = (k,l) \\ c_{ij} & \text{if } (i,j) = (l,k) \\ d_G(i,k) & \text{if } (i,j) \neq (k,l), (i,j) \neq (l,k) \end{cases}$$

The new demands are $q(s_{ij}) = q(s_{ji}) = \frac{1}{2}q_{ij}$.

The same example that as was previously shown is given in order to analyse the transformation proposed by Longo, Aragão and Uchoa, see Figure 1.5. Additionally, Table 1.2 provides the distances between the pairs of nodes.

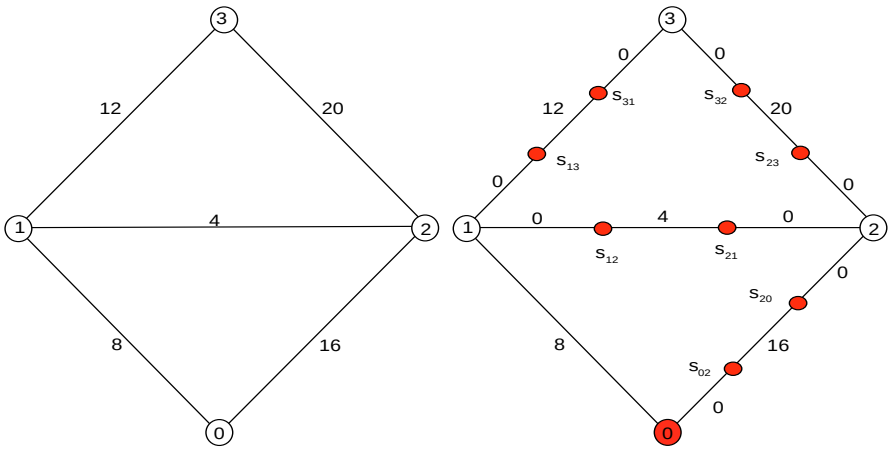


Figure 1.5: Longo, Aragão and Uchoa's Undirected Graph

	0	s_{02}	s_{20}	s_{12}	s_{21}	s_{13}	s_{31}	s_{23}	s_{32}
0	-	0	12	8	12	8	20	8	20
s_{02}	0	-	16	8	16	8	20	12	20
s_{20}	12	16	-	4	0	4	16	0	16
s_{12}	8	8	4	-	4	0	12	4	12
s_{21}	12	16	0	4	-	4	16	0	16
s_{13}	8	8	4	0	4	-	12	4	12
s_{31}	20	20	16	12	16	12	-	16	0
s_{23}	8	12	0	4	0	4	16	-	20
s_{32}	20	20	16	12	16	12	0	20	-

Table 1.2: Distance matrix according to Longo, Aragão and Uchoa's Transformation for an undirected graph

Longo, Aragão and Uchoa's transformation establishes the equivalence of the original arc routing problem and its node routing counterpart in which an UCARP with $|E_R|$ required edges results in a SCVRP with $2|E_R| + 1$ nodes. In order to solve the problem, they introduce the additional restrictions that a previously known set of $|E_R|$ pairwise disconnected edges must belong to every solution, and that nodes s_{ij} and s_{ji} must be visited in sequence. These two sequences are either from s_{ij} to s_{ji}, which means that the edge (i, j) is traversed in the same order as in the CARP solution, or from s_{ji} to s_{ij}, which means that the edge (j, i) is traversed as in the same order in the CARP solution.

They developed a code that is basically an adaptation of a robust branch-and-cut-and-price algorithm[1] for the CVRP, which restricts the paths to those visiting vertices s_{ij} and s_{ji} in sequence. They showed that the branch-and-cut-and-price algorithm yields an effective way of

[1]In Chapter 2, exact algorithms are described in more detail.

attacking the CARP, since it is significantly better than the exact methods created specifically for that problem.

Baldacci and Maniezzo's Transformation

Let $G = (V, E)$ be an undirected graph. Baldacci and Maniezzo (see [7]) proposed a transformation of the UCARP to the SCVRP that replaces each required edge with two nodes. As in Longo, Aragão and Uchoa's transformation, only two side nodes s_{ij} and s_{ji} are associated for each edge $(i, j) \in E_R$. The resulting CVRP instance is defined on the complete undirected graph $\hat{G} = (\hat{V}, \hat{E})$, where $\hat{V} = \bigcup_{(i,j)\in E_R}\{s_{ij}, s_{ji}\} \bigcup \{0\}$.

The distance function is defined as follows[2] :

$$d_{\hat{G}}(0, s_{ij}) = d_G(0, i) + \frac{1}{2}c_{ij}$$

$$d_{\hat{G}}(s_{ij}, s_{kl}) = \begin{cases} 0 & \text{if } (i,j) = (l,k) \\ \frac{1}{2}c_{ij} + d_G(i,k) + \frac{1}{2}c_{kl} & \text{otherwise} \end{cases} \tag{1.4.9}$$

The new demands are $q(s_{ij}) = q(s_{ji}) = \frac{1}{2}q_{ij}$.

Again, Figure 1.6 shows the previous example and the transformation proposed by Baldacci and Maniezzo, and Table 1.3 provides the distances between the pairs of nodes.

[2]Actually, Baldacci and Maniezzo generalized the expression (1.4.9) as follows:

$$\hat{c}_{\hat{G}}(s_{ij}, s_{kl}) = \begin{cases} -UB + (1-2\beta)c_{ij} & \text{if } (i,j) = (l,k) \\ \beta c_{ij} + d_G(i,k) + \beta c_{kl} & \text{otherwise} \end{cases}$$

with $0 \leq \beta \leq \frac{1}{2}$ but they always set β to $\frac{1}{2}$ and $-UB = 0$ for simplicity, where UB is the cost of a known feasible solution of the CARP instance. They generalized the expression in order to prove the correctness of the transformation described in broad terms.

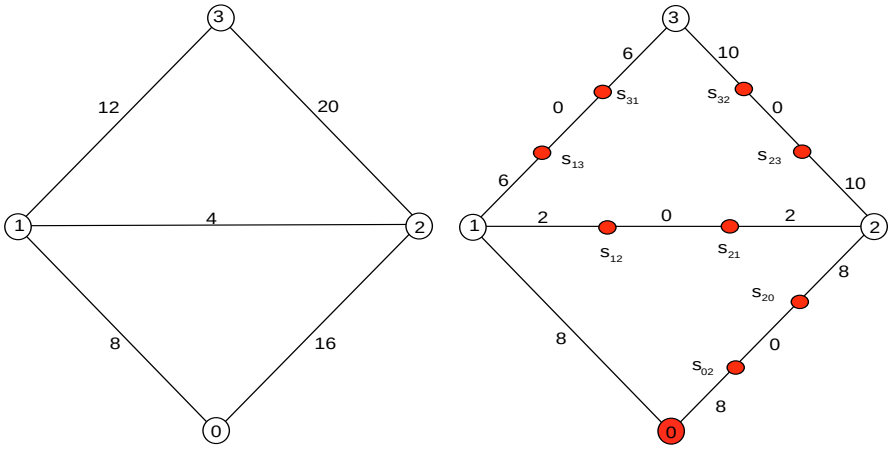


Figure 1.6: Baldacci and Maniezzo's Undirected Graph

	0	s_{02}	s_{20}	s_{12}	s_{21}	s_{13}	s_{31}	s_{23}	s_{32}
0	-	8	20	10	14	14	26	22	30
s_{02}	8	-	0	18	22	22	34	30	38
s_{20}	20	0	-	14	10	18	30	18	34
s_{12}	10	18	14	-	0	8	20	16	24
s_{21}	14	22	10	0	-	12	24	12	28
s_{13}	14	22	18	8	12	-	0	20	28
s_{31}	26	34	30	20	24	0	-	32	16
s_{23}	22	30	18	16	12	20	32	-	0
s_{32}	30	38	34	24	28	28	16	0	-

Table 1.3: Distance matrix according to Baldacci and Maniezzo's Transformation for an undirected graph

If $|E_R|$ is the number of required CARP edges, then Baldacci and Maniezzo's transformation reduces the UCARP instance into an equivalent SCVRP instance that has $2|E_R|+1$ nodes. They propose the elimination of the $-UB$ values and the explicit imposition into the solution of every edge connecting the two nodes associated to required edges. That is, for each required edge, the corresponding cost is set to zero instead of to $-UB$, and additional constraints are added to the formulation of the CVRP to force the $|E_R|$ edges corresponding to the UCARP required edges to be in the optimal solution, since it is no longer a general CVRP, but a more general node routing problem instance with additional constraints added.

Baldacci and Maniezzo developed a branch-and-cut algorithm[3] to optimally solve the transformed UCARP instances.

Finally, it should be borne in mind that Longo, Aragão and Uchoa's transformation is a particular case of Baldacci and Maniezzo's transformation setting $\beta = 0$ and $-UB = 0$.

Maniezzo and Roffilli's Transformation

Let $G = (V, A)$ be a directed graph. Maniezzo and Roffilli (see [95]) proposed a transformation of the DCARP to the ACVRP that replaces each required arc with one node. Each arc $(i, j) \in A_R$ is therefore only associated with one node, which we will denote by m_{ij}. Bear in mind that we will keep the notation introduced by Pearn, Assad and Golden, even if the node m_{ij} is not situated in the middle of the arc in the current

[3] In Chapter 2, exact algorithms are described in more detail.

transformation. The resulting CVRP instance is defined on the complete directed graph $\hat{G} = (\hat{V}, \hat{A})$, where $\hat{V} = \bigcup_{(i,j)\in A_R}\{m_{ij}\}\bigcup\{0\}$.

The distance function is defined as follows:

$$d_{\hat{G}}(0, m_{ij}) = d_G(0, i)$$

$$d_{\hat{G}}(m_{ij}, m_{kl}) = c_{ij} + d_G(j, k)$$

The new demands remain unchanged: $q(m_{ij}) = q_{ij}$.

To illustrate the transformation by Maniezzo and Roffilli see Figure 1.7. The instance contains 4 nodes and 5 arcs, where $V = \{0, 1, 2, 3\}$ is the set of vertices, $A = \{(0, 1), (2, 0), (1, 2), (1, 3), (3, 2)\}$ is the arc set and $A_R = \{(2, 0), (1, 2), (1, 3), (3, 2)\}$ is the set of required arcs. Table 1.4 provides the distances between pairs of nodes for the example depicted in Figure 1.7.

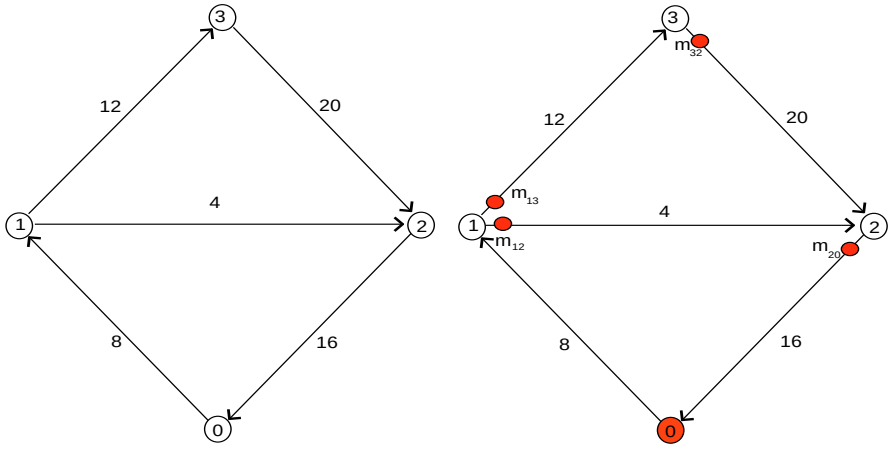


Figure 1.7: Maniezzo and Roffilli's Directed Graph

	0	m_{12}	m_{13}	m_{20}	m_{32}
0	-	8	8	12	20
m_{12}	20	-	28	4	40
m_{13}	48	56	-	32	12
m_{20}	16	24	24	-	36
m_{32}	36	44	44	20	-

Table 1.4: Distance matrix according to Maniezzo and Roffilli's Transformation for a directed graph

This transformation takes advantage of the need to work on a directed graph and uses a modified approach which transforms a DCARP instance into an ACVRP instance, by the simple means of: associating an ACVRP node which inherits the arc demand with each DCARP required arc; maintaining the depot node; and redefining distances according the previous distance function. This results in a graph with $|A_R| + 1$ nodes that defines an instance of ACVRP as having the solution set which can be mapped 1 to 1 into equivalent DCARP solutions.

Maniezzo and Roffilli used this transformation to solve a real-world problem consisting of a waste collection problem. The transformation of the problem into a node routing equivalent was shown and it was tested under various approximative algorithms, both from the literature instances and real-world instances.

A Generalized Transformation

The new transformation for the CARP that is proposed in this section, holds well for undirected, directed and mixed arc routing problems, and

can be viewed as a generalization of the aforementioned CARP transformations. The transformation is described as follows.

In order to transform a CARP into a CVRP, each required arc or edge is replaced with one node or two nodes, respectively. Let $G = (V, A \cup E)$ be a directed, undirected or mixed graph, and let only one node, called middle node, m_{ij}, be associated with each $(i, j) \in A_R$, and let two nodes, called side nodes, s_{ij} and s_{ji}, be associated with each $(i, j) \in E_R$. The resulting CVRP instance is defined on the complete graph $\hat{G} = (\hat{V}, \hat{A} \cup \hat{E})$, where $\hat{V} = \bigcup_{(i,j)\in A_R}\{m_{ij}\} \bigcup_{(i,j)\in E_R}\{s_{ij}, s_{ji}\} \bigcup\{0\}$.

To define the distances between nodes in the CVRP, one can consider of the new nodes as being placed along the original arc or edge (i, j) at equal "spacings" from their original nodes. Therefore, the distance function can be defined as follows:

$$d_{\hat{G}}(0, v) = \frac{c_{ij}}{2} + d_G(0, i) \text{ if } v = s_{ij} \text{ or } v = m_{ij}$$

$$d_{\hat{G}}(m_{ij}, v) = \begin{cases} \frac{c_{ij}}{2} + d_G(j, k) + \frac{c_{kl}}{2} & \text{if } v = s_{kl} \text{ or } v = m_{kl} \\ \frac{c_{ij}}{2} + d_G(j, 0) & \text{if } v = 0 \end{cases}$$

$$d_{\hat{G}}(s_{ij}, v) = \begin{cases} \frac{c_{ij}}{2} + d_G(i, k) + \frac{c_{kl}}{2} & \text{if } v = s_{kl} \text{ or } v = m_{kl} \\ \frac{c_{ij}}{2} + d_G(i, 0) & \text{if } v = 0 \\ 0 & \text{if } v = s_{ji} \end{cases}$$

The new demands are $q(s_{ij}) = q(s_{ji}) = \frac{1}{2}q_{ij}$ and $q(m_{ij}) = q_{ij}$.

Two examples are now given to show this generalized transformation. In the undirected case, the generalized transformation coincides with Baldacci and Maniezzo's transformation, see Figure 1.6 and Table 1.3.

In the directed case, Figure 1.8 illustrates the same instance that was previously shown in Maniezzo and Rofilli's transformation with 4

nodes and 5 arcs, where 4 of these arcs are required. Table 1.5 provides the distances between pairs of nodes for the generalized transformation in the directed case.

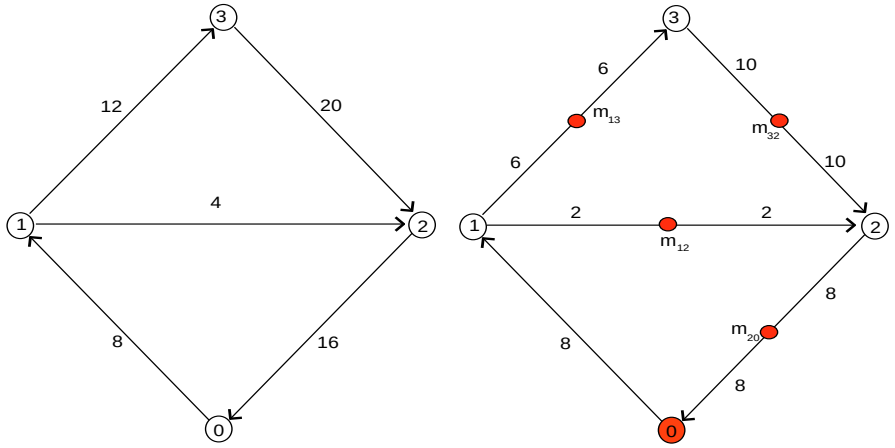


Figure 1.8: Generalized transformation for a Directed Graph

	0	m_{12}	m_{13}	m_{20}	m_{32}
0	-	10	14	20	30
m_{12}	18	-	32	10	48
m_{13}	42	52	-	34	16
m_{20}	8	18	22	-	38
m_{32}	26	36	40	18	-

Table 1.5: Distance matrix for a directed graph

For an illustration of the generalized transformation for mixed graphs, see Figure 1.9. The instance under consideration contains 4 nodes, 3 arcs and 2 edges, where $V = \{0, 1, 2, 3\}$ is the set of vertices, $A = \{(1, 0), (1, 3), (3, 2)\}$ is the arc set, $E = \{(2, 0), (1, 2)\}$ is the edge set

and $A_R = \{(1,3),(3,2)\}$ and $E_R = \{(2,0),(1,2)\}$, are the sets of required arcs or edges, respectively. Table 1.6 provides the distances between the pairs of nodes.

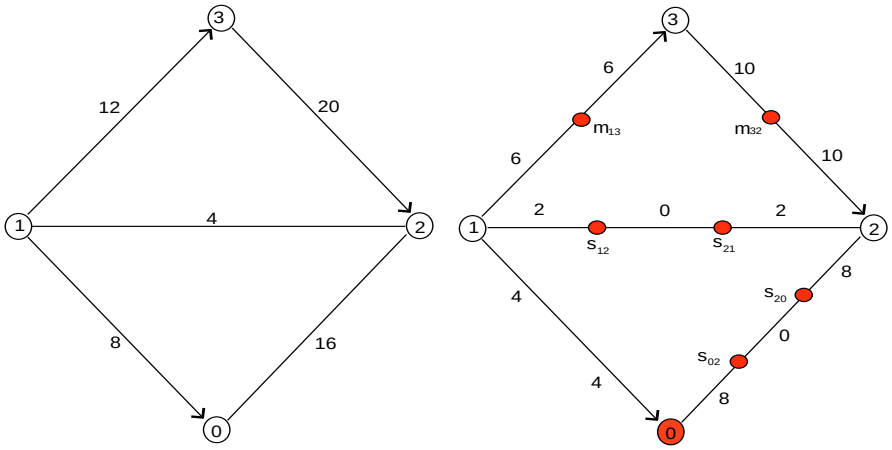


Figure 1.9: Generalized transformation for a Mixed Graph

	0	s_{02}	s_{20}	s_{12}	s_{21}	m_{13}	m_{32}
0	-	8	24	22	18	26	42
s_{02}	8	-	0	30	26	34	50
s_{20}	20	0	-	14	10	18	34
s_{12}	10	18	14	-	0	8	24
s_{21}	14	26	10	0	-	12	28
m_{13}	38	46	34	32	28	-	16
m_{32}	22	30	18	16	12	20	-

Table 1.6: Distance matrix for a mixed graph

To sum up, we propose this transformation, which is valid not only for undirected graphs, but also for directed and mixed graphs. This transformation can be seen as a generalization all of the transformations

that have been proposed in the literature. Note that for undirected graphs, if $|E_R|$ is the number of required edges, then this transformation reduces the UCARP instance into an equivalent SCVRP instance, which has $2|E_R|+1$ nodes. For directed graphs, if $|A_R|$ is the number of required arcs, this transformation reduces the DCARP instance into an equivalent ACVRP instance, which has $|A_R| + 1$ nodes. For mixed graphs, this transformation reduces the MCARP instance into an equivalent ACVRP instance, which has $2|E_R| + |A_R| + 1$ nodes.

The main reason for the proposal of this general transformation valid for undirected, directed and mixed graphs is that the vast majority of real-world problems in the field of routing are represented by mixed or directed graphs, for instance, in a road network that represents a city, there are both two-way streets and one-way streets.

Computational Results for the Generalized Transformation

Concluding remarks about the generalized transformation are now discussed.

In the UCARP, the generalized transformation coincides with Baldacci and Maniezzo's transformation, when setting $\beta = \frac{1}{2}$ and $-UB = 0$. In this case, the validity of the transformation requires no testing since this has been carried out in Baldacci and Maniezzo (see [7]).

For the MCARP, no transformation can be found in the literature, and the generalized transformation proposed here constitutes the first transformation that allows mixed graphs to be handled. No comparison

have been performed to test the validity of our transformation due to the lack of previous work on this topic.

For the DCARP, we propose a transformation that differs from Maniezzo and Roffilli's transformation because we prefer to consider the new nodes as being placed along the original arc (i, j) at equal "spacings" and not at the beginning of the arc. By placing the nodes in the middle of the arc, we are also providing information on the previous and the subsequent weight of the arc, and hence *a priori* information is known about two required arcs: the arc being left and the arc being entered. In Maniezzo and Roffilli's transformation case, information is available about the required arc being left, but not about the required arc being entered. In order to test the validity of the generalized transformation for directed graphs, we perform a comparison against Maniezzo and Roffilli's transformation. In Tables 1.7 and 1.8, the results from computational experiments are presented and are performed on an hp Pavilion dm4 (Intel $CORE^{TM}$ i5 - 430M processor 2.26 GHz and 2GB RAM) with CPLEX 12.2 using exact algorithms. The set of instances are shown in Tables 1.7 and 1.8 containing DeArmon and Benavent instances adapted to the DCARP instances. Those instances were first presented in [7]. From among the Belenguer instances, we have only selected a subset, denoted by A, that has the least number of vehicles and the biggest capacity. The set of instances denoted by B, C and D are the same instances as in the set denoted by A, but the number of vehicles and their capacity have been varied. Therefore, in order to prevent CPLEX from running out of

memory while it is solving the model, we have selected only the A subset.

The results from the computational tests performed are presented in order to evaluate the quality of the transformation proposed for DCARP instances.

Columns 6 and 8 from Tables 1.7 and 1.8 show the optimum value solving DCARP instances as ACVRP instances using Maniezzo and Roffilli's transformation (MRT) and the generalized transformation (GT), respectively. Columns 7 and 9 show the CPU times spent by CPLEX in solving the instances using Maniezzo (CPU_{MTR}) and Roffilli's transformation and the generalized transformation (CPU_{GT}), respectively. The CPU time is given in seconds.

The CPLEX branch-and-bound algorithm for solving these problems uses modern features such as cutting planes and heuristics to find integer solutions[4]. CPLEX has many parameters that allow users to customize the way the CPLEX branch-and-bound algorithm operates. While this variety of parameters provides various ways to improve performance, a user cannot realistically experiment with all the possible combinations of parameters settings. Since we are not interested in performance improvement, combinations of parameter settings are not employed here.

[4]In Chapter 2, exact and approximative algorithms are described in more detail.

DeArmon	$\|V\|$	$\|A_R\|$	K	Q	MRT	CPU_{MRT}	GT	CPU_{GT}
1	12	22	5	5	316	0.44	316	0.56
2	12	26	6	5	339	0.2	339	0.34
3	12	22	5	5	275	0.06	275	0.02
4	11	19	4	5	287	0.33	287	0.11
5	13	26	6	5	377	0.09	377	0.09
6	12	22	5	5	298	0.44	298	0.19
7	12	22	5	5	325	0.55	325	0.55
10	27	46	1	270	256	1.01	256	0.67
11	27	51	1	270	261	1.12	261	0.64
12	12	25	4	10	304	20.76	304	4.62
14	13	23	7	35	458	0.11	458	0.36
15	10	28	6	41	548	602.35	548	183.89
16	7	21	5	21	100	0.06	100	0.08
17	7	21	4	37	58	0.05	58	0.05
18	8	28	5	24	127	0.2	127	1.12
19	8	28	5	41	91	0.16	91	0.17
20	9	36	5	37	164	0.22	164	0.22
21	10	11	3	27	55	0.02	55	0.02
22	11	22	4	27	121	2.28	121	0.91
23	11	33	6	27	156	3.67	156	4.91
24	11	44	8	27	200	103.99	200	15.44
25	11	55	1	270	225	1.33	225	0.38
Mean						33.61		9.79

Table 1.7: Comparison on the DeArmon instances

Belenguer	$\|V\|$	$\|A_R\|$	K	Q	MRT	CPU_{MRT}	GT	CPU_{GT}
1A	24	39	2	200	173	0.36	173	0.22
2A	24	34	2	180	227	1.34	227	3.37
3A	24	35	2	80	81	0.28	81	0.3
4A	41	69	3	225	400	1080.00	400	69.00
5A	34	65	3	220	423	334.97	423	90.89
6A	31	50	3	160	223	11.39	223	2.03
7A	40	66	3	200	279	3.84	279	4.37
8A	30	63	3	200	386	95.00	386	93.68
9A	50	92	3	235	323	34.96	323	9.7
10A	50	97	3	250	428	8.61	428	153.98
Mean						157.08		42.75

Table 1.8: Comparison on subset A of Belenguer instances

In view of the results, it is worth mentioning that by using the generalized transformation, in 10 out of 22 instances from DeArmon the CPU is better than in Maniezzo and Roffilli's transformation and 5 out of 22 instances result in a draw. Furthermore, in 6 out of 10 instances from Belenguer, the CPU time using the generalized transformation is better than in Maniezzo and Roffilli's transformation[5].

Regarding the mean of the CPU time in the two sets of benchmark instances, that obtained using Maniezzo and Roffilli's transformation is more than three times greater than in the generalized transformation. This could be due to the possibility that the generalized transformation converges more quickly than does Maniezzo and Roffilli's transformation thanks to the placement of the new nodes in the graph which help in

[5]In Table 1.7, problems DeArmon 10, 11 and 25 have 10 vehicles with capacity 27. In order to simplify the problem only one vehicle with capacity 270 was considered to prevent that CPLEX from running out of memory while solving these problems.

the branch-and-bound algorithm and the cutting planes. As we have previously mentioned, through placing the nodes in the middle of the arc, we are also providing information on the previous and the subsequent weight of the arc, and hence *a priori* information is known about two required arcs: the are being left and the arc being entered. In the Maniezzo and Roffilli's transformation case, information is available about the required arc being left but not about the required arc being entered.

1.5 Multi-Objective Routing Problems

Up to this point, routing problems have been covered where only a single objective is taking into account. The goal of this section is to draw an overview of how multi-objective optimization can bring to routing problems.

According to [78], *Multi-Objective Routing Problems* are mainly used in three ways:

(i) To extend classic academic problems in order to improve their practical application (while never losing sight of the initial objective). In this context, the problem definition remains unchanged, and new objectives are added.

(ii) To generalize classic problems by adding objectives instead of one or several constraints and/or parameters. Note that, in the literature, this strategy has notably been applied to the routing problem with

time window constraints where the time windows are replaced by one or several objectives.

(iii) To study real-life cases in which the objectives have been clearly identified by the decision-maker and are dedicated to a specific practical problem or application.

The objectives taken into account when solving a multi-objective routing problem can be multiple and diverse. The minimization of the cost of the solution generated is the most common objective. Cost can be expressed in many ways, such as the distance travelled, the time required, the number of customers visited, and even the economical cost.

Other commonly used objectives are related to the balance of the routes. Those objectives include the makespan, as given in the studies by Corberán *et al.* [32], Pacheco and Martí [108], Lacomme *et al.* [85], and Reiter and Gutjaht [118], among others. Tour balancing could also be introduced by minimizing the difference between the length of the longest tour and the length of the shortest tour, see Jozefowiez *et al.* [77]. Further objectives related to tours exists, such as the minimization of the risk in the case of hazardous transportation problems, and the maintenance of a tour in a cluster.

In the literature, many papers deal with vehicle routing problems with time windows in which additional objectives are considered. These objectives include the minimization of the lateness or earliness to the bounds of the windows and/or the number of violated constraints, see, for instance, Baràn and Schaerer [9], and Castri-Gutierrez *et al.* [26]. Other

objectives are linked to assigning priority to certain customers in a effort to visit those with the highest priorities, see Park and Koelling [109] and [110]. It is also possible to define economic objectives, such as increasing customer satisfaction, as in Sessomboon *et al.* [124], and improving the customer-driver relationship, as in Ribeiro and Lourenço [119]. Another family of objectives deals with optimizing the access to the visited nodes by means of forming a set of unvisited nodes, see Bowerman *et al.* [21].

Other objectives considered in the literature are related to the management of the fleet. The most common objectives are the following: minimizing the size, see [32] and [108]; optimizing the effectiveness of the vehicle utilization, see El-Sherbeny [46] and Sessomboon *et al.* [124]. Objectives can also be related to that being transported: consideration of passengers, see Corberán *et al.* [32] and Pacheco and Martí [108]; the prevention of the deterioration of perishable products, see Park and Koelling [109] and [110].

Before closing this section, it should be emphasized that, in a multi-objective optimization problem, there is more than one conflicting objective, and hence there is no single optimal solution as in single-objective optimization but rather there exist a number of solutions which are all optimal. Without any further information, no solution from the set of optimal solutions can be said to be better than any other. Since a number of solutions are optimal, in a multi-objective optimization problem, many such optimal solutions are important and are known as Pareto-optimal solutions. Since none of these solutions can be said to be any

better than the others, the first task in a multi-objective optimization problem is to find as many such Pareto-optimal solutions as possible. This constitutes the fundamental difference between a single objective and a multi-objective optimization problem.

In a multi-objective optimization problem, a solution x dominates a solution y, written as $x \prec y$ if x is at least as good as y for each objective. As a consequence, x is a *non-dominated solution* if there is no solution y that dominates x. According to this logic, the set P^* of all non-dominated solutions is called the *non-dominated set*, *Efficient set*, *Pareto-optimal set*, or *Pareto-optimal front*.

Chapter 2

Algorithms and a new GRASP for Routing Problems

2.1 Introduction

Routing problems are very popular combinatorial optimization problems that can generally be considered as problems that require searching for the best solution from among a large number of finite discrete candidate solutions.

Traditionally, routing problems are addressed as single-objective optimization problems although, as it is well known, most are multi-objective in nature in which several objectives must often be satisfied at the same time despite being in conflict with each other.

In this chapter, first a short review of techniques capable of solving routing problems is given. These can be broadly categorized into the following two groups: *exact methods* and *approximate methods*[1]. Exact

[1]Hereinafter, *algorithm* and *method* will be used indifferently.

algorithms yield an optimal solution to the problem, given a sufficient (normally a huge) amount of time. The main limitation of exact algorithms is that they are highly expensive in terms of computing time, even for medium-sized problems (50-100 nodes). Hence, approximate methods (heuristics and metaheuristics) are introduced. Approximate algorithms are developed to find the optimal solution or a feasible solution near the optimal solution in a reasonable computing time. Section 2.2.1 briefly highlights certain well-used exact algorithms to solve routing problems, and Section 2.2.2 describes some well-known approximate algorithms.

In this dissertation, a parallel is drawn between single-objective optimization problems finding optimal or feasible solutions, with multi-objective optimization problems finding non-dominated or Pareto optimal solutions. Therefore, once exact and approximate algorithms are described, several performance assessments, depending on whether it is a single-objective optimization problem or a multi-objective optimization problem can be determined. In Section 2.3, *performance metrics* are used to provide useful information on how efficient and effective a method is, i.e., for measuring their quality.

The overall aim of this thesis is to solve a real-world waste collection problem whose size is very large. The impossibility of solving this problem using exact algorithms leads us to use approximate algorithms. To face the difficulty of the considered problem, a Greedy Randomized Adaptive Search Procedure (GRASP) and a Variable Neighbourhood Descent (VND) is proposed in this chapter. GRASP is an approximate method that

has been successfully applied to a wide variety of complex real problems and has demonstrated great of flexibility in its adaptation to various optimization problems, see Section 2.4. The first step towards the aim of solving the real-world problem involves dealing with a simplified version of the problem. This simplified version is a DCARP, and includes similar features to the real problem. In this chapter, the simplified version of the problem is solved by designing an algorithm. In fact, two different approaches are considered: in Section 2.4.1, a single-objective version of the GRASP is presented in which the objective is to minimize the total routing cost; and in Section 2.4.2, a bi-objective version of the GRASP is described in which a second objective function is introduced, which balances the routes.

2.2 Solution Methods

In general, two different types of solution methods can be distinguished for solving optimization problems. These are exact algorithms, yielding the optimal solution although consuming too much time, and approximate algorithms, yielding near-optimal solutions within shorter computation times, which are more appropriate for solving real-world problems.

2.2.1 Exact Methods

Exact algorithms guarantee finding the optimal solution for the problem given a sufficient amount of time. In general, the most successful of these algorithms strive to reduce the solution space and/or the number of alternatives that require examination in order to reach the optimum solution.

A number of well-known exact methods used in the field of routing are briefly described; namely, dynamic programming, branch-and-bound, branch-and-cut, branch-and-prize, and branch-and-cut-and-prize algorithms.

• **Dynamic Programming (DP).** This method uses a process of simplifying a complicated problem by recursively dividing it into smaller subproblems (also known as *stages*). In each stage a decision is required. Each stage also has a number of *states* associated with it. The decision at one stage transforms the current state into the state in the following stage. Hence, the decision updates the state for the next stage. The decision to move to the next state is only dependent on the current state, not on previous states or decisions. This is the fundamental dynamic programming principle of optimality. It means that the problem can be broken into smaller independent problems. Only certain routing problems are amenable to DP. Dijkstra's shortest path and Floyd-Warshall's shortest path algorithms are classic problems in which DP can be applied.

In the field of VRPs, DP was first proposed by Eilon, Watson-Gandy and Christofides in 1971 (see [43]) by considering a CVRP with a fixed number of vehicles. However, in the field of CARPs, DP are usually based on matching techniques (see [62]).

• **Branch-and-Bound (BB).** This is an optimization technique based on the systematic search of all possible solutions, while discarding a large number of non-promising candidate solutions. The algorithm for a routing problem can be described as follows. First a *lower bound* (assuming a minimization problem) is obtained, by solving a Linear Programming (LP) relaxation of the Integer Linear Programming (ILP) by dropping the condition that all variables have to be integer. In the case where the obtained solution is an integer, the optimal solution to the original problem is found. Otherwise, a branch and bound tree is built. The *branching tool* splits a given set of candidates into two smaller sets and the two new linear problems are then solved with an additional constraint; either an upper or a lower bound on one of the variables which are supposed to be integer but because of the relaxation are associated with a real value in the current solution. In the subsequent iterations, each solution serves as new candidate in the tree. With the branching tool, the number of problems grows exponentially. In order to prevent this exponential growth, a *bounding tool* is introduced, which computes an upper bound for the function to be optimized within a given subset. The process of discarding fruitless solutions is usually called *pruning*. The algorithm stops when all nodes of the search tree are either pruned or

solved.

In 1981 Christofides *et al.* (see [29]) solved VRPs using a BB scheme. Then, in 1992 (see [89]) Laporte, *et al.* used a BB algorithm to solve the asymmetrical VRPs. On the other hand, the first attempt to solve CARP exactly was by Hirabayashi *et al.* in 1992. They used a BB (see [75] for further details).

• **Branch-and-Cut (BC).** The BC algorithm uses a BB technique with an additional cutting step. The idea is to reduce the search space of feasible candidates by adding new constraints (cuts). In the *cutting plane algorithm* (or Row Generation), first the LP relaxation of the original ILP is solved by dropping the condition that all variables have to be integers, and in the case where the obtained solution is not integer, a *cut* is generated that separates the optimal solution from the true feasible set. A valid cut has two properties: any feasible point of the ILP satisfies the cut, and the current optimal solution to the LP will violate the cut (see [131]). In contrast to BB, in the BC only a subset of the original constraints are considered in the LP relaxation. It is unusual for all constraint families of exponential size to be included. If the solution of the LP relaxation is feasible for ILP, then it is an optimal solution; otherwise the separation algorithms will check if at least one constraint is violated by the solution of the LP relaxation. In the case where none of the omitted constraints are violated, an optimal solution to the LP is found. If at least one violated constraint has been detected by a separation procedure, then the violated constraint(s) is (are) added to the LP, and the updated LP is solved again.

This is repeated until the separation procedures fail to detect additional violated constraints. If this does not terminate with an optimal solution to ILP, then the problem is decomposed into two new problems as is carried out in BB.

There are numerous papers that solve VRPs using BC algorithms, to name just a few, see, for instance, [1], [94], and [105]. For ARPs, Belenguer and Benavent (see [14]) presented a cutting plane algorithm to solve the CARP, which is partly based on several classes of valid inequalities presented by the same authors (see [13]).

• **Branch-and-Prize (BP).** This algorithm is similar to BC, except that the procedure focuses on Column Generation rather than on Row Generation. In fact, Pricing and Cutting are complementary procedures for the tightening of a LP relaxation. In BP, columns are left out of the LP relaxation since there are too many columns to be efficiently handled and most of these columns have their associated variable equal to zero in an optimal solution anyway. In order to check the optimality of a LP solution, a subproblem, called the *pricing problem*, is solved in an attempt to identify columns with a profitable reduced cost. If such columns are found, the LP is reoptimized. Branching occurs when no profitable columns are found, and the LP solution does not satisfy the integrality conditions. The BP algorithm applies column generation at every node of the BB tree.

BP algorithms for different variants of VRPs are considered by Salani, see [120]. For ARPs, BP techniques can be found, for instance in

[48] and [30].

• **Branch-and-Cut-and-Prize (BCP).** This algorithm is a combination of the BC and BP algorithms. Very good results were obtained by Fukasawa *et al.*, see [55], by applying BCP to the VRP. An exact solution using a BCP algorithm was suggested for CARP by Letchford and Oukil (see [90]).

As previously stated, the main limitation of exact methods is the amount of time they spend on solving problems of a considerable size. In order to solve this problem, approximate algorithms are introduced below.

2.2.2 Approximate Methods

Given the aforementioned limitations of exact methods in solving large-sized problems, approximate methods are often preferred. Furthermore, in most practical situations, approximate methods are adopted since they present the only way to address the problem that ensures high-quality solutions.

Approximate algorithms, like heuristics and metaheuristics, are techniques that almost solve difficult problems in a reasonable amount of time. However, there is no guarantee that the obtained solution is optimal, or that the same quality of solution will be obtained every time the algorithm is applied. The justification of this situation is clear: approximate algorithms are non-deterministic algorithms, i.e., algorithms that can exhibit different behaviours on different runs, as opposed to an exact algorithm, also known as a deterministic algorithm.

Heuristics

The term *heuristic* is related to the task of solving problems in an intelligent way using the information available. The term heuristics comes from the Greek word *heuriskein* and means to discover or to find out, which, in turn, is derived from *eureka*, the famous exclamation of Archimedes on discovering the beginning.

According to [98], "*a heuristic is described as a procedure that has a high degree of confidence to find high-quality solutions with reasonable computational cost, while there is no guarantee of optimality or feasibility, and even, in some cases no means to establish how good the solution is. Heuristic, using the adjective as opposed to exact . . .*"

Therefore, in contrast to the exact methods, in heuristic methods the speed of the process is as important as the quality of the solution.

A heuristic method can be used in the following cases:

- When there is no an exact method to solve the problem.
- When exact methods to solve the problem exist, but their use is computationally expensive or impractical.
- When the heuristic method is more flexible than an exact method, allowing, for example, the modelling of difficult conditions to be incorporated.

There are various types of heuristics, making it difficult to give a complete and unique classification. According to Semet and Laporte (see

[123]), heuristic methods in the field of routing problems can be classified into three categories:

• **Constructive heuristics.** These gradually build a feasible solution, by striving to optimize the objectives. In the field of routing problems, one or more routes are built by inserting, at each iteration, one or more customers (nodes, arcs or edges) in one of the routes under construction. Routes can be constructed using *sequential methods* (one route at a time is built) or *parallel methods* (all routes are built simultaneously). In sequential methods, at each iteration, the best unvisited customer is chosen and is inserted into the best position in the current route without considering whether it would be better to insert it in other routes. Each route ends when no more customers can be inserted, because a constraint would be violated. In parallel methods, routes are built choosing, at each iteration, the best unvisited customer and inserting this customer in the best position of the best route from among all the routes. Parallel methods usually obtain better solutions than those obtained by sequential methods, although their computation costs are higher.

The Nearest Neighbor algorithm and the Clarke and Wright algorithm, see [31], perhaps constitutes the most widely known heuristic for VRPs. Mole and Jameson, see [102], and Christofides, Mingozzi and Toth, see [28], suggest two representative sequential algorithms applied to problems with an unspecified number of vehicles.

In the ARPs counterpart, some well-known algorithms are: the

Construct-Strike algorithm, the Path-Scanning algorithm, the Augment-Merge algorithm, and the Parallel-Insert algorithm. To go into more detail, or examine more algorithms of a constructive nature, see [72].

• **Two-phase heuristics.** In the field of routing, these heuristics are divided into two classes: *cluster-first, route-second* and *route-first, cluster-second* methods. In the first case, customers (nodes, arcs or edges) are first assigned into feasible clusters, and a vehicle route is constructed for each cluster. In the second case, only one tour is built with all customers and then this giant tour is segmented into feasible vehicle routes. Route-first, cluster-second methods were first put forward by Beasley in 1983 (see [11]).

Three elementary clustering methods in VRPs are the sweep algorithm (see for instance [58]), the Fisher and Jaikumar algorithm (see [50]), and the Bramel and Simchi-Levi algorithm (see [22]).

Three ARP heuristics in this category are the Ulusoy-Partitioning algorithm, the Cut algorithm, and the Cycle-Assignment algorithm. For details see [72].

• **Improvement heuristics or Local Search heuristics.** The basic idea behind a local search, also known as neighbourhood search, is to look for better solutions around a given point in the search space. An algorithm of this kind starts with a candidate solution and then, iteratively, moves from this point to a neighbouring solution where the objective which is being optimized has been improved. Therefore, unlike previous methods, local search heuristics start with a solution of the problem and try to

improve it gradually. The method ends when no affordable solution can be found that improves the previous solution.

In order to upgrade the feasible solution, a sequence of customer exchanges within or between vehicle routes (*intra-route, inter-routes*) is executed. Popular intra-route improvements include λ-*opt* (remove λ edges or arcs from the tour and reconnect the tour) and *Or-opt* (relocate sequences of one, two and three nodes in all possible positions). Some inter-route improvements are now described. The *relocate operator* moves a node from one route to another, the *exchange operator* exchanges two nodes in different routes, the λ-*interchange* consists of exchanging customers from one route to another one as long as this remains feasible, where λ represents the maximum number of customers to be exchanged. The *2-Opt** strives to combine the two routes without changing the orientation of the tours, by appending the last customers of the second route after the first customers of the first route.

The principal limitation of heuristic methods lie in their inability to find a global optimum (see Figure 2.1), mainly due to the fact that these algorithms fail to use any mechanism that allows them to continue the search for the optimal in the case of being trapped in local optimum. To solve this problem, we introduce other, more intelligent, search algorithms (called metaheuristics) that prevent, as far as possible, this problem. Such algorithms are procedures that guide known high-level heuristics, and prevent them from being trapped in a local optimum.

The main goal of metaheuristic algorithms therefore, is to "escape"

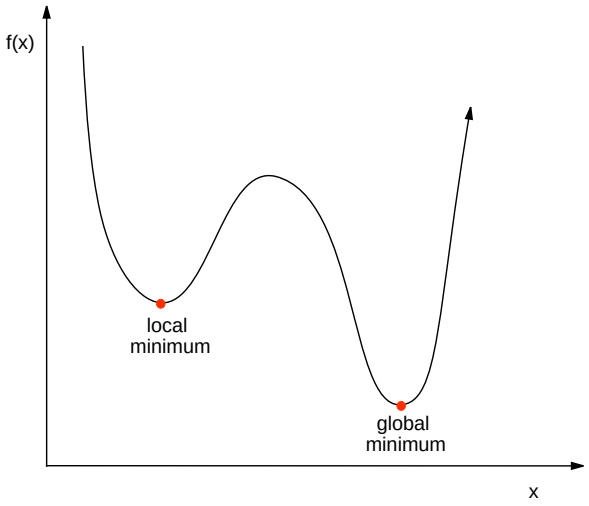


Figure 2.1: Local minimum and global minimum

from the current "local optimum" by exploring a larger portion of the feasible region through the incorporation of diversification tools, as explained below.

Metaheuristics

The suffix *meta* means *beyond, in an upper level*. Metaheuristics are strategies for the design or improvement of heuristics in order to obtain high-performance solutions. In essence, a metaheuristic is a top-level strategy that guides an underlying heuristic in the solution of a given problem. The term metaheuristic was introduced by Fred Glover in 1986, see [60], but it is only in the last few years that certain researchers in the field have tried to establish a definition. One possible definition is that

presented in [79], which can be stated as follows:

"*A metaheuristic is an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search spaces using learning strategies to structure information in order to find efficiently near-optimal solutions.*"

It could therefore be said that metaheuristic algorithms are iterative heuristic approaches that present two essential qualities: to encourage the discovery of better solutions in the search space by focusing on good solutions and improving upon them (intensification); and to encourage the exploration of the solution space by broadening the focus of the search into new areas (diversification). These two qualities are both complementary and necessary. A purely intensification-based search cannot accept poorer solutions and therefore cannot escape from local optimum; a purely diversification-based search has no quality control by which to reject poorer solutions and achieve good results.

The classification of metaheuristics presents a difficult task mainly since real implementations cannot extract a set of features that are completely homogeneous and cannot allow complete separation between these features and other metaheuristics. There are metaheuristic approaches that are called population-based algorithms (diversification ability), such as genetic algorithms or ant colony optimization, and there are metaheuristic approaches that are known as trajectory-based algorithms (intensification ability) based on a number of local search neighbourhoods, such as tabu search, variable neighbourhood search, and simulated annealing. The

following algorithms remain quite popular in the field of routing.

• **Genetic Algorithms (GA).** These algorithms are based on an analogy with the evolution of a population and hence they mimic the evolution process in nature. This idea dates back to the 1950's, although the theoretical foundation of GAs was established by John Holland in 1975 (see [76]), and GAs then became popular as an intelligent optimization technique that can be adopted for the solution of many difficult problems.

Genetic Algorithms work on a population of individuals representing possible solutions to a given problem. In traditional GAs, each individual is usually represented by a string of bits analogous to *chromosomes* and *genes*: the parameters of the problem are the genes that are joined together in a solution chromosome. A *fitness value* is assigned to each individual in order to judge their ability to survive and breed. The highly fit individuals are given a chance to breed by being selected for *reproduction*. Thus, the selection process usually favours the fittest individuals. Good individuals may be selected several times in one iteration, while poor individuals may not be selected at all. By selecting the fittest individuals, favourable characteristics spread throughout the population over several generations, and the most promising areas of the search space are explored. Finally, the population should converge towards an optimal or near-optimal solution in a single-objective optimization problem and towards the Pareto optimal solutions in the multi-objective optimization problem.

During the reproduction phase of a GA, two individuals breed by

combining their genes in an operation called crossover. Not all selected pairs undergo a *crossover*. A random choice is applied, where the likelihood of crossover is some given probability. If crossover is not performed, offspring are produced simply by duplicating their parents. Crossover allows the basic genetic material of parents to pass to their children, who then form the next generation. Another operation that is performed by GAs is *mutation*, which is applied to each child generated from a crossover. With a certain small probability, each gene may be altered. Thus, crossover allows a rapid exploration of the search space by producing large jumps, while mutation allows a small amount of random search.

There are many papers that solve VRPs using GAs, see, for example [4]. For multi-objective VRPs, see [106] or [116] and for multi-objective ARPs, see [85].

• **Ant Colony Optimization (ACO).** This algorithm, first proposed by Dorigo *et al.* in 1991 (see [38]), is a population-based metaheuristic inspired by the foraging behaviour of ants, which enables them to find the shortest path between the nest and a food source. While walking from food sources to the nest and vice versa, ants deposit a substance called pheromone on the ground. When they decide which direction take, they choose, with higher probability, those paths that are marked by stronger *pheromone* concentrations. This basic behaviour is the basis for a cooperative interaction which leads to the emergence of shortest paths.

The are three steps in ACO algorithms. In the first step each ant constructs a solution from the set of feasible problem solutions, by adding

one solution component in each step of the construction process to the current partial solution. In the second optional step, a certain problem-specific action may be required, which cannot usually be performed by a single ant. For example, a local search may be applied to optimize the set of generated solutions. The new optimized solutions are then used to decide which pheromone trails to update. Finally, a global pheromone (or update process) is performed at the end of each iteration, where updating the pheromone values depends on the quality of the generated solutions in the current iteration. This is usually performed by decreasing the pheromone value for all solutions, in a process called evaporation, and increasing the pheromone value of good solutions. Evaporation is a tool that ACO uses to explore new areas of the search space and to prevent being trapped in local optimum.

References to ACO algorithms in the field of routing include [15], [9], and [121].

• **Simulated Annealing (SA).** This algorithm, proposed by Kirkpatrick *et al.* in 1983 (see [80]), is a well-known metaheuristic search method. The denomination is adopted from the annealing of solids, where the aim is to minimize the energy of the system by using slow cooling until the atoms reach a stable state. The slow cooling technique allows atoms of the metal to line themselves up and to form a regular crystalline structure that has high density and low energy. The initial temperature and the rate at which the temperature is reduced is called the *annealing schedule*.

The basic algorithm of SA may be described as follows. Starting

from a certain feasible solution of the problem, an attempt is made to optimize this solution by using a method analogous to the annealing of solids. A neighbour of this solution is generated using an appropriate method, and the cost of the new solution is calculated. If the new solution is better than the current solution in terms of reducing cost, then the new solution is accepted, otherwise, the new solution is accepted in accordance with a certain probability. The probability of acceptance is usually set to $e^{-\Delta/T}$, where Δ is the change in cost between the old and the new solution, and T is the current *temperature*. The probability of acceptance thus decreases exponentially with the badness of the move. The annealing temperature is first chosen to be high so that the probability of acceptance will also be high, and almost all new solutions are accepted. The temperature is then gradually reduced so that the probability of acceptance of low quality solutions becomes very small; in this way, high temperatures allow a better exploration of the search space, while lower temperatures allow the fine tuning of a good solution. The process is repeated until the temperature approaches zero or no further improvement can be achieved.

For some interesting references on SA, see [46], [92], and [80].

• **Variable Neighbourhood Search (VNS).** This algorithm was proposed by Hansen and Mladenović, see [69] and [70], and is another local search-based metaheuristic which exploits many different neighbourhoods, in order to escape a local optimum. The main idea is based on exploring the search space of the current solution by gradually increasing the neighbourhood size, within which a new solution is generated, until a certain

stopping condition is reached. Furthermore, whenever a new solution is generated in the current neighbourhood, a local search is applied to this solution to optimize it before the replacement decision is made.

In the basic VNS algorithm, the neighbourhood size increases from 1 to a certain maximum value (a parameter). The most important step in the algorithm is the *shaking step* in which a new solution is generated within the current neighbourhood. It is crucial to choose a shaking procedure that will allow enough perturbation of the solution, while preserving, at the same time, the most favourable solution features which should be utilized in obtaining a near-optimal solution. On the other hand, the *local search step* in the VNS algorithm intensifies the search to obtain a local optimum, which may replace the current solution. In the acceptance criterion, called *move or not step*, if the local optimum is better than the current solution, then move is adopted there and the search continues until the stopping condition is reached.

A well-known variant of the VNS is the Variable Neighbourhood Descent (VND) where the best neighbour of the current solution is considered instead of a random solution, i.e., the shaking step is omitted. The local search metaheuristic that will be designed for the problem of interest in this dissertation will be that of the VNS and is explained in greater detail in the following section.

References in the field of routing on the VNS include [83] and [73].

• **Tabu Search (TS).** This is another popular search technique proposed by Glover in 1977 (see [59]). Its name is derived from the word "taboo"

meaning forbidden or restricted. The principle of TS is to encourage a more diverse search by remembering solutions which have been recently explored and excluding them from the current search. Tabu Search uses a short-term memory called a *tabu list*, in which all moves that have been recently performed during the search are recorded. Moves in the tabu list are considered prohibited by the search and cannot be performed again for a certain number of iterations. This concept is introduced in order to prevent cycles.

The size of the tabu list is usually fixed, so that a number of old tabu moves can be removed to allow new, recently performed, moves to be recorded and the duration of a move that has been declared tabu is called its *tabu tenure*. Hence, the structure of the neighbourhood being searched varies dynamically from one iteration to another. However, always restricting the search to non-tabu moves may prevent some promising search areas from being explored. To rectify this problem, TS often makes use of *aspiration criteria*, which allow a solution to be accepted if it improves the best solution found so far.

It is also sometimes fruitful in TS to make use of an intensification and/or a diversification mechanism. As was previously mentioned, intensification strives to enhance the search around good solutions, while diversification strives to force the algorithm to explore new search areas, in order to escape local optimum.

See, for instance, [108] and [23] for studies on the TS algorithm in the VRP and in the ARP.

The last metaheuristic is now briefly described and is explained in greater detail in the following section. This method is selected to solve the real-world waste collection problem, which constitutes the focus of interest of this dissertation since it has been successfully applied to a wide variety of complex real problems. Furthermore, the following algorithm presents an interesting quality: its way of combining abilities of intensification and of diversification.

• **Greedy Randomized Adaptive Search Procedure (GRASP).** GRASP was first proposed by Feo and Resende in 1995 (see [49]). A GRASP is a multi-start, or iterative, process, in which each iteration consists of two phases: a randomized construction phase and a local search phase. The construction phase builds a feasible solution, whose neighbourhood is investigated until a local optimum is found during the local search phase, and then the best overall solution is kept as the result. Elements to be added in the construction phase are determined by ordering all elements in a candidate list with respect to a greedy function that estimates the benefit of selecting each element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, which is not always the top candidate.

In the field of routing, the GRASP has been widely used, see, for instance [5] and [129].

To conclude, it is worth mentioning that the majority of metaheuristics are usually too dependent on the specific problem, or a high knowledge of the problem is required. Furthermore, the metaheuristic convergence

and/or the solution quality cannot be ensured. However, the experimental behaviour of most metaheuristics is extraordinary, making the only feasible alternative to find a quality solution in reasonable time for many intractable problems. In general, metaheuristics behave as very robust and efficient methods.

Today, the majority of researchers studying metaheuristic approaches use metaheuristics that work together in one way or another. There is currently great interest and projection in building hybrid metaheuristics, hyperheuristics, and matheuristics.

A **hybrid algorithm** is a technique that results from the combination of a metaheuristic with other optimization techniques, i.e., with exact algorithms, heuristics or metaheuristics. For a literature review see [18] and [117], where a classification of hybrid algorithms can also be found. The motivation behind such hybridizations of different algorithmic concepts is usually to obtain a better performance of systems that exploit and unite advantages of the individual pure strategies.

Hyperheuristics are very high-level strategies, even more than metaheuristic strategies. A hyperheuristic can be viewed as a heuristic that iteratively chooses between given heuristics or metaheuristics in order to solve an optimization problem (see [25]). The basic idea underpinning the hyperheuristic is to use population heuristics and/or metaheuristics acting on a particular problem, so that at any moment the hyperheuristic determine the best method for solving the problem, and changes the state of the problem. The most important motivation for using hyperheuristics

is that the same hyperheuristic method can be applied to a range of problems, thereby solving the limitation presented in heuristics and metaheuristics of being too dependent on the problem.

Matheuristics are algorithms made by interoperation of metaheuristics and mathematic programming techniques. An essential feature is the exploitation in some part of the algorithms of features derived from the mathematical model of the problems of interest. Matheuristics show how it is possible combine both techniques, either using mathematical programming to improve or design metaheuristics or using metaheuristics to improve known mathematic programming techniques. For a literature review see [96] and for a survey on matheuristics in the field of routing problems and transportation logistics, see [37].

2.3 Performance Metrics

With the rapid increase in the number of techniques available to address optimization problems in general and routing problems in particular, the relevance of performance assessment has grown. As with single-objective optimization, in multi-objective optimization the notion of performance involves both the quality of the solution found and the time to generate such a solution. Hence, when a new and innovative methodology is presented to solve any optimization problem, then useful information on how well a method is must be provided. To this end, performance metrics are explained in order to assess the quality of the new method against other existing methods.

Performance metrics differ depending on whether the proposed algorithm is designed for the solution of a single-objective optimization problem or a multi-objective optimization problem. In contrast with single-objective problems, where one can compare the best-known solutions, multi-objective problems have whole sets of solutions to compare with at least two aims: to minimize the distance from the generated solutions (called the Pareto approximation, approximation set, non-dominated set, or non-dominated solutions) to the true Pareto front, and to maximize their diversity, i.e. the coverage of the Pareto front. A description of the metrics to be used in this dissertation for both single-objective and multi-objective optimization problems now follows. It is worth mentioning that there exist many more metrics than those given below.

2.3.1 Metrics for Single-Objective Problems

The comparison of the performance of solution methods is always a difficult task since it involves solution quality, computational effort, robustness, and other factors. There are various alternatives to measure the quality of an algorithm. One easy way is to set a limit of iterations or to set the computing time and compare the objective function values. The best method is then judged to be that with the best objective value. Another alternative is to set the value of the objective function and compare the computing time or the number of iterations needed to reach the value. Yet another way is to count the number of times that the algorithm is able to attain the optimal solution (in the case where this is known) or

the best-known solution found in the literature.

Due to its popularity in the field of routing, we will use a performance measure known as *gap*, which, for each problem, assesses the percentage deviation of the objective function value obtained using the new method and that obtained using other methods:

$$gap = \frac{f(x) - f^*(x)}{f^*(x)} \cdot 100$$

where $f(x)$ represents the value of the objective function obtained using the new algorithm, and $f^*(x)$ represents the best-known value of the objective function. It should be borne in mind that $f^*(x)$ could be: the optimal solution, if it is known; a lower bound of the algorithm or, as assumed in this work, the best approximate solution found in the literature. We assume that all feasible solutions have a positive value and that the objective function is a minimization function.

2.3.2 Metrics for Multi-objective problems

In multi-objective optimization, performance assessments are more complicated than in single-objective optimization since the outcome of the optimization problem is seldom a single solution but a set of trade-off. We apply the indicators and guidelines proposed in Knowles *et al.* [81]. In particular, they propose the following four evaluators as the most discriminant indicators to compare multi-objective methods.

• **Coverage Metric.** The coverage metric, $\mathcal{C}(A, B)$, was proposed by Zitzler in 1999 (see [132]), and calculates the proportion of solutions in

the estimated efficient frontier B, which are weakly dominated[2] by the efficient solutions in the estimated frontier A:

$$\mathcal{C}(A,B) = \frac{|\{b \in B | \exists a \in A : a \preceq b\}|}{|B|}.$$

The metric value $\mathcal{C}(A,B) = 1$ means all decision vectors in B are weakly dominated by A. On the other hand, $\mathcal{C}(A,B) = 0$ means that no solution of B is weakly dominated by A. Note that $\mathcal{C}(A,B)$ is not necessarily equal to $1 - \mathcal{C}(B,A)$.

• **Hypervolume Indicator.** This metric, first introduced by Zitzler and Thiele in 1998 (see [133]), calculates the size of the space covered, that is, it computes the hypervolume of the portion of the objective space that is weakly dominated by an approximation set A. In order to measure the hypervolume, the objective space must be bounded by fixing a reference set R that could be either the true Pareto front, if known, or a approximation set from the literature. Henceforth, R will be computed by merging all solutions found by all the different algorithms into one set and by storing all non-dominated solutions. A variation of this measure, computed as difference in the hypervolume to a set reference R is considered, i.e., $I_H(R) - I_H(A)$. The smaller value corresponds to higher quality, in contrast to the original hypervolume.

In Figure 2.2, the hypervolume measures the size of the dominated region, bounded by some reference point. The figure on the left shows the hypervolume of the efficient solutions obtained by algorithm A, while the figure on the right shows the hypervolume of the efficient solutions

[2] $a \preceq b$ means that a weakly dominates b, i.e., a is not worse than b in any of the objectives.

obtained by algorithm B. The light-blue areas in the two figures represent the hypervolume differences. In the light of the figures, we can conclude that algorithm B is better than algorithm A, by considering the hypervolume indicator.

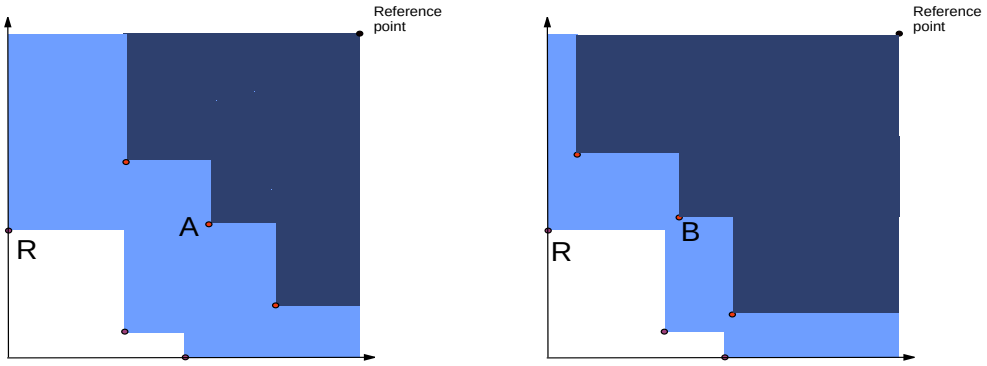


Figure 2.2: Hypervolume indicator

• **Unary Epsilon Indicator.** This metric, first introduced by Zitzler *et al.* in 2003 (see [134]), makes direct use of Pareto dominance, and hence is highly intuitive. For two approximation sets A and B, the *binary epsilon indicator*, denoted by $I_\epsilon(A, B)$, measures the minimum factor ϵ by which each point in the set B can be multiplied such that the resulting transformed approximation set is weakly dominated by A. This binary indicator is transformed into a *unary indicator* by fixing the set compared, B. This set is then called reference set, R. The unary epsilon indicator, denoted by $I_\epsilon(A)$, calculates the minimum factor ϵ by which each point of R can be multiplied such that the resulting transformed approximation set is weakly dominated by A. A lower indicator value

indicates a (possibly) better approximation set. This indicator relies on the ϵ-dominance relation, $\preceq_\epsilon$, defined as:

$$\mathbf{x^1} \preceq_\epsilon \mathbf{x^2} \Longleftrightarrow \forall i \in 1, \ldots, m : f_i(x^1) \leq \epsilon \cdot f_i(x^2)$$

• **R Indicator Family.** The R indicators, proposed by Hansen and Jaszkiewicz in 1998 (see [68]), can be used to assess and compare Pareto set approximations on the basis of the set of utility functions that represent the preferences of the decision maker. Now, suppose that the decision maker's preferences are given in terms of a parameterized utility function u_λ and a corresponding set Λ of parameters. In our case, the u_λ function represents the weighted Tchebychev function but it could be another utility function, such as, the weighted sum. In particular, we will use the binary I_{R_2} indicator, defined as follows:

$$I_{R_2}(A, B) = \frac{\sum_{\lambda \in \Lambda} u^*(\lambda, A) - u^*(\lambda, B)}{|\Lambda|}$$

where u^* is the maximum value reached by the utility function u_λ with weight vector λ on a Pareto set approximation A. In our case, that is $u^*(\lambda, A) = \min_{x \in A}\{\max_{i \in 1 \ldots m} \lambda_i |f_i(x) - f_i^*(x)|\}$. Similar to the epsilon indicators, the R indicators can be defined as unary or binary versions by replacing one set with an arbitrary, but fixed reference set R; $I_{R_2}(A) = I_{R_2}(R, A)$. The indicator values are to be minimized.

The performance assessment tool suite is provided in PISA. Therefore, to accomplish this task, the four metrics are computed using the package available on the PISA website[3].

[3]http://www.tik.ee.ethz.ch/pisa

This tool first performs a postprocessing stage in which the lower and upper bounds of the objective vectors[4] are calculated, all the values are normalized and the non-dominated front of all objective vectors provided are then computed, considering all algorithms to attain the reference set, R. Once this initial step has been performed, the tool computes the four metrics.

Note that each indicator is based on different preference information. Therefore, using all the indicators will provide more information than simply using just one indicator.

2.4 A GRASP for Routing Problems

A GRASP (see [49]) is a memory-less multi-start metaheuristic, where each iteration consists of two phases: a construction phase and a local search phase. The most distinguishing characteristic of a GRASP is the way in which it combines greediness with randomness in the construction phase, in which initial solutions are built by adding one element at a time.

Let x be the partial solution under construction in a given iteration, and let $\mathcal{C}$ be the candidate set with all the remaining elements that can be added to x. Without loss of generality, it is then assumed that the problem is a minimization problem. The GRASP construction uses a greedy function $g(c)$ to measure the contribution of each candidate element $c \in \mathcal{C}$ to the partial solution x, i.e., it calculates the cost of selecting

[4]It is worth mentioning that the metrics are only provided for those problems with more than one solution in every algorithm; this constraint is stipulated by the software itself.

each element. To this end, a *restricted candidate list* (RCL) is built by selecting a subset of all elements in a greedy fashion. The RCL contains those elements whose incorporation into the partially built solution would yield the smallest increase in the objective function value, f, i.e., it is the subset of candidate elements from $\mathcal{C}$ with good evaluations according to g. In particular, if g_{min} and g_{max} are the minimum and the maximum evaluations of g on $\mathcal{C}$ respectively, then $RCL = \{c \in \mathcal{C} | g(c) \leq g_{min} + \alpha(g_{max} - g_{min})\}$, where $\alpha \in [0, 1]$. From the RCL, an element is then selected at random, after which the RCL is updated to reflect the fact that a new element has been added to the solution and is no longer available for selection. Selection of an element and update of the RCL are repeated until a complete solution has been built. From this solution, a local search phase starts until a local optimum is found.

Note that the size of the RCL, α, is a parameter of the GRASP that controls the balance between greediness and randomness. If α is small, the search is relatively greedy. If α is large, it is relatively random. In the extreme cases, setting $\alpha = 0$ leads to a completely deterministic greedy search, whereas setting $\alpha = 1$ entails a completely random search.

In the local search, the neighbourhood of the initial solution is explored. The solutions generated by a GRASP construction are not guaranteed to be locally optimal. Hence, it is almost always beneficial to apply a local search to attempt to improve each solution constructed. A local search algorithm works in an iterative fashion by successively

replacing the current solution with a better solution from its neighbourhood. It terminates when there are no better solutions in the neighbourhood.

The best solution over all the GRASP iterations is returned as the result. Success for a GRASP depends on an efficient neighbourhood-search technique as well as a good starting solution provided by the construction phase.

To sum up, a GRASP can be seen as a metaheuristic which captures the advantages of pure greedy algorithms (intensification) and also of random construction procedures (diversification). Pseudo-code for a generic GRASP for a minimization problem is given in Algorithm 1.

The stopping criterion for the GRASP is usually that the elapsed CPU time is greater than a given limit, or that the number of iterations is greater than a given limit.

Once the GRASP has been explained in broad terms, we will explain in detail a new GRASP for routing problems. Let $G = (V, A)$ be a graph representing the road network of the problem. The GRASP will be designed for two different versions of the problem, the single-objective version, in which the objective is to minimize the total cost of all routes, and the multi-objective version, in which two objectives are considered: to minimize the total cost of all routes and to minimize the cost of the route that has the maximum cost in order to get a more balanced set of routes. In both versions of the problem, K routes, r_k, $k = 1, \ldots, K$, will be obtained as solution. Note that routes are composed of nodes. Routes, $r = \bigcup_{k=1}^{K} r_k$, will visit all nodes V in the graph $G = (V, A)$ in which a

distinguished vertex 0 is the depot where vehicles are located.

Algorithm 1 GRASP

1: $f^* \leftarrow \infty$
2: **while** stopping criterion is not satisfied **do**
3: $x \leftarrow \emptyset$;
4: Compute $\mathcal{C}$ with the candidate elements that can be added to x;
5: **while** $\mathcal{C} \neq \emptyset$ **do**
6: For all $c \in \mathcal{C}$ compute $g(c)$, $g_{min} = \min_{c \in \mathcal{C}} g(c)$ and $g_{max} = \max_{c \in \mathcal{C}} g(c)$;
7: Define $RCL \leftarrow \{c \in \mathcal{C} | g(c) \leq g_{min} + \alpha(g_{max} - g_{min})\}$ with $\alpha \in [0, 1]$;
8: Select c^* at random from $RCL(\mathcal{C})$;
9: Add c^* to partial solution $x \leftarrow x \cup \{c^*\}$;
10: Update $\mathcal{C}$ with the candidate elements that can be added to x;
11: **end while**
12: **if** x is infeasible **then**
13: Apply a repair procedure to make x feasible;
14: **end if**
15: $x \leftarrow LocalSearch(x)$;
16: **if** $f(x) < f(x^*)$ **then**
17: $x^* \leftarrow x$; $f^* \leftarrow f(x)$;
18: **end if**
19: **end while**
20: **return** x^*

2.4.1 Single-Objective GRASP

Construction Phase

In order to construct solutions according to the GRASP, two construction algorithms to build the routes will be describe so that the best route can be selected. Both algorithms are designed to optimize the objective function, f, which minimizes the total cost of the K routes. For the first construction algorithm, we implement a standard procedure based on the

Mole and Jameson insertion heuristic (see [102]). Specifically, we use a particular case of the algorithm known as the extra cost heuristic. For the second construction algorithm, we implement a more elaborate algorithm called the regret heuristic (see [112]). The proposed solution is the best between the two solutions obtained.

The *extra-cost function* computes the change in the objective function incurred by inserting node j at the position, i, in route k. We select the minimum extra cost value and the node is inserted into the best possible route and position. The *regret function* computes the difference in the cost of inserting the node j into its second-best route and into its best route at the best positions. We select the maximum regret value and the node is inserted in the best possible route at the best position.

We have proposed two different constructive methods since the extra-cost algorithm presents an obvious problem in postponing the placement of difficult nodes (those with high extra-cost values) until the last iterations where we have very little freedom of action to insert them. The regret heuristic strives to circumvent the problem by incorporating look-ahead information when selecting the node to insert.

The two constructive algorithms will be explained in detail. Note that the proposed construction algorithms are parallel algorithms since all routes are built simultaneously either using the first algorithm, the extra cost heuristic, or using the second algorithm, the regret heuristic.

- **Step 1** (compute a list of seeds). Choose K *seed nodes* to initialize each vehicle route, r_k, $k = 1, \ldots, K$. The seeds are chosen as those

nodes hardest to visit. In our case that means that: the first seed will be the node with the highest cost from the depot; the second seed will be node with the highest cost from the depot and from the first seed; the third seed will be the node with the highest cost from the depot, the first seed and the second seed; and so on. In an intermediate step, vertex $j^* \in V \backslash (S \cup \{0\})$ is chosen to be added to the set of seeds, S, from a set of candidates with greater values of the function $g^{(0)}(j) = \sum_{s \in S \cup \{0\}} \min\{d_G(s,j), d_G(j,s)\}$. It is worth mentioning that we select the minimum between the two distances because we are dealing with an asymmetric graph. To build the RCL, we compute $g^{(0)}_{min} = \min_{j \in V \backslash (S \cup \{0\})} g^{(0)}(j)$ and $g^{(0)}_{max} = \max_{j \in V \backslash (S \cup \{0\})} g^{(0)}(j)$.

- **Step 2** (Assigning nodes to routes according to the extra-cost function). Starting with the set of non-selected nodes, J, initially composed of all nodes except those of depot and the seeds and in an intermediate step, vertex $j^* \in J$ is chosen to be added to route r_k at the position i in a greedy fashion from the subset of candidate elements, i.e., from the RCL, with lower values according the function $g^{(1)}(j,k,i) = c_{r_k(i-1)j} + c_{jr_k(i)} - c_{r_k(i-1)r_k(i)}$, where $r_k(i)$ denotes the vertex of the i-th position at route r_k. Note that if the capacity of a vehicle is exceeded, then no more nodes can be allocated to this route. To build the RCL, we compute $g^{(1)}_{min} = \min_{\substack{j \in J \\ k=1,\ldots,K \\ i=1,\ldots,|r_k|}} g^{(1)}(j,k,i)$

and $g^{(1)}_{max} = \max_{\substack{j \in J \\ k=1,\ldots,K \\ i=1,\ldots,|r_k|}} g^{(1)}(j,k,i)$.

- **Step 3** (Assigning nodes to routes according to the regret function). Starting again with the set of non-selected nodes, J, in an intermediate step, vertex $j^* \in J$ is chosen to be added to route r_k at the position i from RCL with greater values according the function $g^{(2)}(j) = g^{(1)}(j,k^2,i_{k^2}) - g^{(1)}(j,k^1,i_{k^1})$, where k^1 denotes the route with the lowest extra cost, k^2 denotes the route with the second-lowest extra cost, and i_k denotes the best position of route k. In mathematical terms, we compute $k^1 = \arg\min_{k=1,\ldots,K} g^{(1)}(j,k,i)$ and $k^2 = \arg\min_{k=1,\ldots,K\setminus\{k^1\}} g^{(1)}(j,k,i)$. Again, if the capacity of a vehicle is exceeded, no more nodes can be allocated to this route. To build the RCL, $g^{(2)}_{min} = \min_{j \in J} g^{(2)}(j)$ and $g^{(2)}_{max} = \max_{j \in J} g^{(2)}(j)$ are computed.

The GRASP construction does not guarantee the feasibility of the solution obtained (although these details have not been introduced into the pseudo-code for simplicity). When this happen a new solution is constructed. We explore the neighbourhood of only feasible initial solutions because it is supposed that these are high-quality or promising solutions and we discard those infeasible solutions obtained in the construction phase since it seems unwise and computationally expensive to explore the neighbourhood of all initial solutions.

The pseudo-code of the GRASP two-phase construction is shown in Algorithm 2.

Algorithm 2 Construction Phase

Input: $G = (V, A)$, K, $\alpha^{(0)}$, $\alpha^{(1)}$, $\alpha^{(2)}$;
Output: $r = (r_k)_{k=1,\dots,K}$ feasible set of routes;
Initialize: $S = \{0\}$, $J = V \setminus \{0\}$, $k \leftarrow 1$, $r \leftarrow \emptyset$, $f^* \leftarrow \infty$;
repeat
    For all $j \in J$ compute $g^{(0)}(j)$;
    Define $RCL^{(0)} \leftarrow \{j \in J | g^{(0)}(j) \geq g^{(0)}_{max} - \alpha^{(0)}(g^{(0)}_{max} - g^{(0)}_{min})\}$;
    Select j^* at random from $RCL^{(0)}(J)$;
    $r_k \leftarrow r_k \cup \{j^*\}$, $S \leftarrow S \cup \{j^*\}$, $J \leftarrow J \setminus \{j^*\}$, $k \leftarrow k + 1$;
until $k = K + 1$;
$r^{(1)} \leftarrow r$, $J^{(1)} \leftarrow J$;
while $J^{(1)} \neq \emptyset$ **do**
    while the capacity of the K vehicles is not exceeded **do**
        For all $j \in J^{(1)}$ compute $g^{(1)}(j, k, i)$;
        Define $RCL^{(1)} \leftarrow \{j \in J^{(1)} | g^{(1)}(j, k, i) \leq g^{(1)}_{min} + \alpha^{(1)}(g^{(1)}_{max} - g^{(1)}_{min})\}$;
        Select j^* at random from $RCL^{(1)}(J^{(1)})$;
        $r^{(1)} \leftarrow r^{(1)} \cup \{j^*\}$, $J^{(1)} \leftarrow J^{(1)} \setminus \{j^*\}$;
    end while
end while
$r^{(2)} \leftarrow r$, $J^{(2)} \leftarrow J$;
while $J^{(2)} \neq \emptyset$ **do**
    while the capacity of the K vehicles is not exceeded **do**
        For all $j \in J^{(2)}$ compute $g^{(2)}(j)$;
        Define $RCL^{(2)} \leftarrow \{j \in J^{(2)} | g^{(2)}(j) \geq g^{(2)}_{max} - \alpha^{(2)}(g^{(2)}_{max} - g^{(2)}_{min})\}$;
        Select j^* at random from $RCL^{(2)}(J^{(2)})$;
        $r^{(2)} \leftarrow r^{(2)} \cup \{j^*\}$, $J^{(2)} \leftarrow J^{(2)} \setminus \{j^*\}$;
    end while
end while
if $f(r^{(1)}) < f(r^{(2)})$ **then**
    $r \leftarrow r^{(1)}$; $f^* \leftarrow f(r^{(1)})$;
else
    $r \leftarrow r^{(2)}$; $f^* \leftarrow f(r^{(2)})$;
end if
return r

Local Search Phase

The Variable Neighbourhood Descent (VND) search is essentially a simple variant of the VNS algorithm, in which the shaking phase is omitted. Therefore, in contrast with VNS, VND is usually completely deterministic.

In Variable Neighbourhood Descent (VND), several different neighbourhoods are explored in order, typically from the smallest and fastest to evaluate to the largest and slowest neighbourhood. The process iterates over each neighbourhood while improvements are found, performing a local search until a local optimum is found at each neighbourhood. Only strictly better solutions are accepted after each neighbourhood search. In VND, the returned solution is likely to be a local optimum. Therefore, the global optimum is likely to be found within a shorter time than when considering only one neighbourhood.

Our VND algorithm uses five neighbourhood structures, $\mathcal{N}_m$, where $1 \leq m \leq m_{max}$ with $m_{max} = 5$, and these are described below. In our particular case, the neighbourhoods are explored from the largest to the smallest since we prefer to perform more "aggressive" moves at the beginning of the search.

1. **Neighbourhood structure 1 ($\mathcal{N}_1$):** Move a set of nodes from various routes to one other route, see Figure 2.3. We strive to include a set of nodes in unsaturated routes, i.e., where the capacity constraint is not exceeded. First, a route is selected in which a number of nodes can be inserted. It is worth mentioning that the

quantity of nodes to be moved is not fixed *a priori*. The best movement is then performed. This is repeated until all routes have been considered.

Figure 2.3 shows the $\mathcal{N}_1$ movement in which one node from the blue route, one node from the red route, and one node from the pink route are selected to be inserted in the green route.

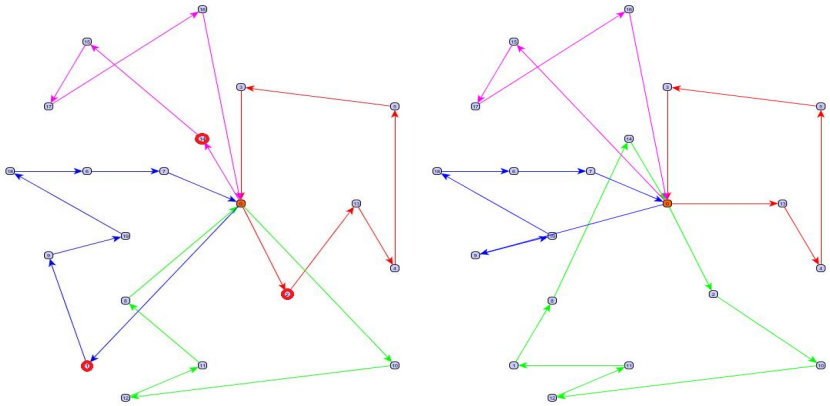


Figure 2.3: $\mathcal{N}_1$

2. **Neighbourhood structure 2 ($\mathcal{N}_2$):** The λ-*interchange* is performed. As previously explained, this consists of swapping λ nodes from one route to another route as long as this is feasible, where λ is the maximum number of nodes to be exchanged. Again, we select one route where the nodes can be inserted. The quantity of nodes that will be moved is not fixed *a priori*, and depends on the quantity of nodes that can be inserted into the selected route. The best movement is then performed. This is repeated until all routes

have been considered.

Figure 2.4 shows the 2-interchange, where two nodes are selected from the red route and are inserted into the green route.

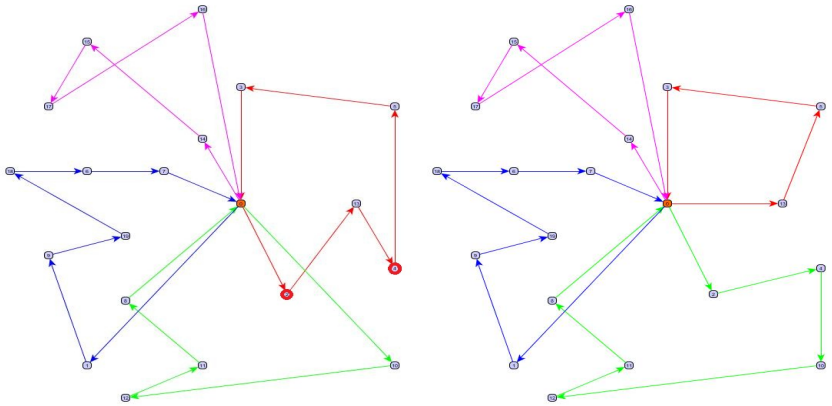


Figure 2.4: $\mathcal{N}_2$ with $\lambda = 2$

Moreover, Figure 2.5 shows the 3-interchange, where three nodes are selected from the blue route and are subsequently inserted into the pink route.

3. **Neighbourhood structure 3 ($\mathcal{N}_3$):** Exchange two nodes between two different routes, i.e., the *exchange operator*, see Figure 2.6. We select all pairs or possible routes and we check which nodes can be exchanged between the routes. This is repeated until all pairs of routes have been considered and the best movement has been performed.

   Figure 2.6 shows the $\mathcal{N}_3$ movement, where one node is selected from the blue route and is inserted in the green route and one node is

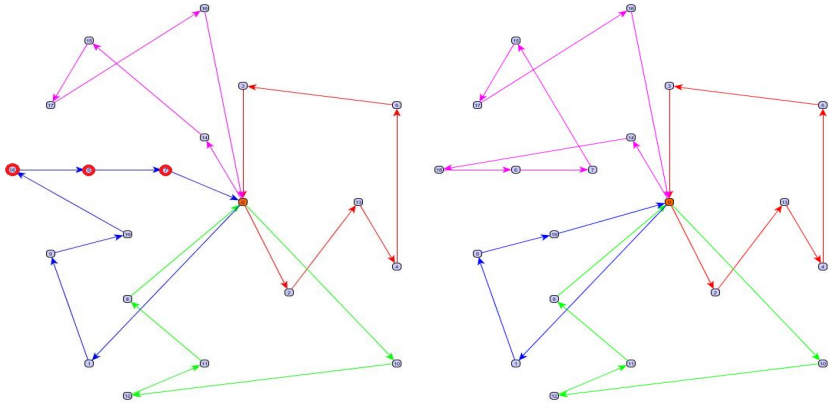


Figure 2.5: $\mathcal{N}_2$ with $\lambda = 3$

selected from the green route and is inserted in the blue route.

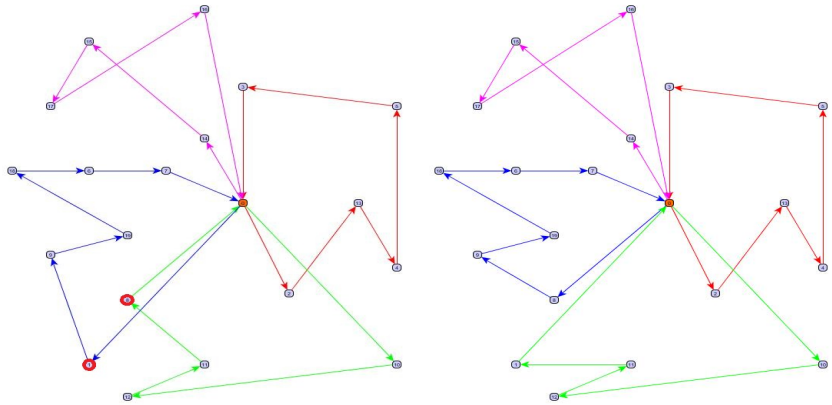


Figure 2.6: $\mathcal{N}_3$

4. **Neighbourhood structure 4 ($\mathcal{N}_4$):** Move a node from one route to another, i.e., the *relocate operator*, see Figure 2.7. We select one route and check which nodes can be inserted from another route. This is repeated until all possibilities have been considered and the

best movement has been performed.

Figure 2.7 shows the $\mathcal{N}_4$ movement, where one node is selected from the green route and is inserted in the blue route.

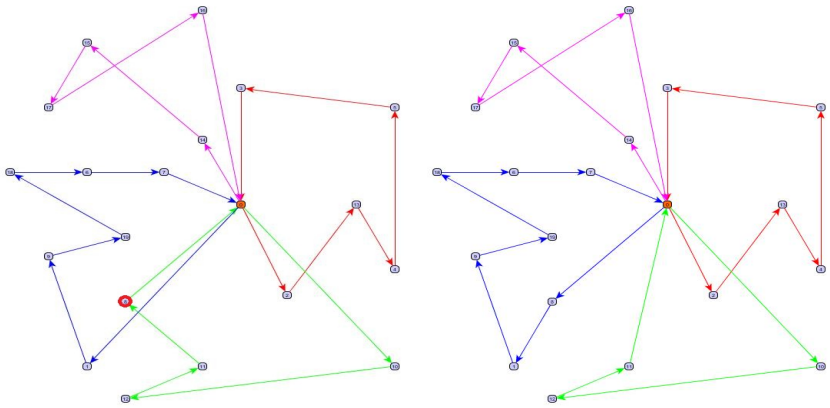


Figure 2.7: $\mathcal{N}_4$

5. **Neighbourhood structure 5 ($\mathcal{N}_5$):** Interchange two consecutive nodes within a route, see Figure 2.8. We select one route and we check all consecutive nodes that can be interchanged within the same route. This is repeated until all possibilities have been considered and the best movement has been performed.

   Figure 2.8 shows the $\mathcal{N}_5$ movement, where two consecutive nodes are selected from the pink route and they exchange positions.

It is worth mentioning again that the local search starts from a given feasible initial solution obtained in the construction phase. To improve the solution, only promising movements are executed, i.e., those whose objective function values are better than the previous objective

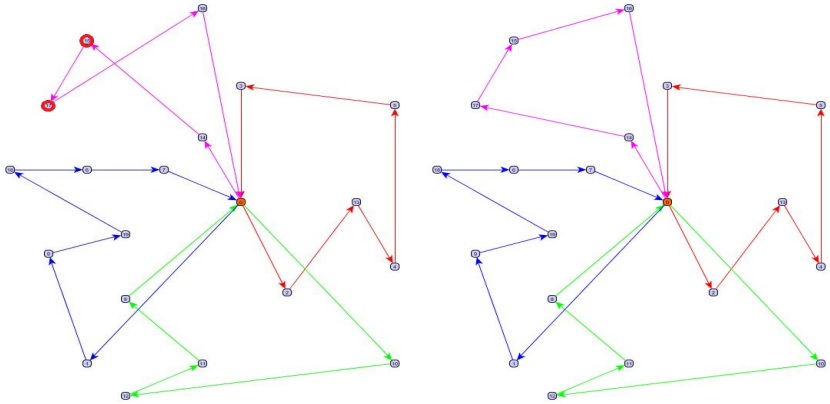


Figure 2.8: $\mathcal{N}_5$

function value. Note that not all movements can be executed, since the resulting solution could be infeasible, that is, one of the constraints could be violated. In this case, this move is simply discarded. For more details, see pseudo-code for the VND given in Algorithm 3.

Algorithm 3 Basic VND

Define the set of neighbourhood structures $\mathcal{N}_m$, $m = 1, \ldots, m_{max}$
Input: r feasible solution;
Initialize: $m \leftarrow 1$;
repeat
    local search: $r' \leftarrow arg\min \mathcal{N}_m(r)$;
    if $f(r') < f(r)$ **then**
        $r \leftarrow r'$ and $m \leftarrow 1$ (centre the search around r and search again in the first neighbourhood);
    else
        $m \leftarrow m + 1$ (switch to another neighbourhood);
    end if
until $m = m_{max}$;
return r

Computational Results

In this section, in order to validate the proposed algorithm, we will test it on a set of benchmark instances and the obtained results are compared against best known competitors. Computational experiments are performed on an Toshiba Portège (Intel Core i7, processor 2.33Ghz and 8GB RAM). The proposed algorithm is implemented using Java and executed using Java OpenJDK 6.

Two sets of instances are considered, taken from [35] and [14]. Both sets of instances are originally undirected CARP instances and Maniezzo and Roffilli in [95] transform these instances into directed CARP instances. Again, the objective is to minimize the total cost. The first set of instances, see Table 2.1, are small-sized problems and contain instances with $7-27$ nodes and from 22 to 110 arcs, of which between 11 to 55 are required. The second file, presented in Table 2.2, contains instances with $24-50$ nodes and from $68-194$ arcs, of which $35-97$ arcs are required.

Since the problems are DCARPs, they will first be transformed into ACVRPs. This task is accomplished by transforming the directed graph as shown in Chapter 1, where two transformations are presented for directed graphs (Maniezzo and Roffilli's Transformation and the Generalized Transformation). The transformation proposed by Maniezzo and Roffilli will be used with the aim of performing a fair comparison between all the algorithms. Furthermore, the well-known Dijkstra's algorithm will be implemented to compute the new cost matrix.

In each experiment of Tables 2.1[5] and 2.2, the total number of nodes, $|V|$, arcs, $|A|$, and required arcs, $|A_R|$, are reported, as are the number of vehicles, K, and the maximum capacity of vehicles, Q. The best known solutions, *best*, are found by Maniezzo and Roffilli in [95], who proposed a modified Iterated Variable Neighbourhood Search by implementing three different shaking strategies, a multi-start algorithm (MS), a genetic algorithm (GA), and a data perturbation procedure (DP). For each instance, the best value; $GRASP$, obtained with the proposed GRASP, the average CPU time in seconds, CPU; and the relative percentage deviation, *gap*, between the obtained solution and the best known solution.

The number of runs and the stopping criterion are parameters of the GRASP. We have employed the same parameters as used in [95] since we are interested in performing a fair comparison. The proposed algorithm is run three times for each problem. The stopping criterion of the algorithm is that either the maximum CPU time of 180 seconds is reached, or the maximum number of iterations: $10,000$. Moreover, the seeds are chosen in the GRASP taking $\alpha^{(0)}$ at random from $[0, 1]$ and in the construction phase, $\alpha^{(1)} = 0$ and $\alpha^{(2)} = 0$, which means, that the greedy function selects the best node instead of one at random from among the best nodes.

[5]Solutions in Table 2.1 for problems DA12 and DA15 marked with an asterisk ($*$) mean that the best solution was modified since, upon double-checking the results (solving them with exact methods) a mistake was found.

Problem	$\|V\|$	$\|A\|$	$\|A_R\|$	K	Q	best	*GRASP*	gap	CPU
DA1	12	44	22	5	5	316	316	0.00%	0.03
DA2	12	52	26	6	5	339	339	0.00%	3.17
DA3	12	44	22	5	5	275	275	0.00%	0.04
DA4	11	38	19	4	5	287	287	0.00%	0.00
DA5	13	52	26	6	5	377	377	0.00%	1.85
DA6	12	44	22	5	5	298	298	0.00%	0.12
DA7	12	44	22	5	5	325	325	0.00%	0.01
DA10	27	54	37	10	27	350	394	12.57%	11.05
DA11	27	102	51	10	27	315	331	5.08%	2.44
DA12	12	50	25	4	10	304*	304	0.00%	0.72
DA14	13	46	23	7	35	458	458	0.00%	0.08
DA15	10	56	28	6	41	548*	556	1.46%	12.09
DA16	7	42	21	5	21	100	100	0.00%	0.01
DA17	7	42	21	4	37	58	58	0.00%	0.00
DA18	8	56	28	5	24	127	127	0.00%	1.59
DA19	8	56	28	5	41	91	91	0.00%	0.01
DA20	9	72	36	5	37	164	164	0.00%	0.30
DA21	10	22	11	3	27	55	55	0.00%	0.00
DA22	10	44	22	4	27	121	121	0.00%	0.20
DA23	11	66	33	6	27	156	156	0.00%	12.13
DA24	11	88	44	8	27	200	200	0.00%	7.82
DA25	12	110	55	10	27	235	239	1.70%	75.61
mean								0.95%	5.88

Table 2.1: Comparison on the DeArmon DCARP instances.

Problem	$\|V\|$	$\|A\|$	$\|A_R\|$	K	Q	best	$GRASP$	gap	CPU
1A	25	78	39	2	200	173	173	0.00%	0.38
1C	25	78	39	3	120	179	182	1.68%	4.16
1C	25	78	39	8	45	248	264	6.45%	27.61
2A	25	68	34	2	180	227	227	0.00%	0.05
2B	25	68	34	3	120	259	259	0.00%	1.97
2C	25	68	34	8	40	457	485	6.13%	49.96
3A	25	70	35	2	80	81	81	0.00%	0.04
3B	25	70	35	3	50	87	87	0.00%	0.92
3C	25	70	35	7	20	140	142	1.43%	5.72
4A	42	138	69	3	225	400	400	0.00%	0.19
4B	42	138	69	4	140	418	**414**	**-0.96%**	24.09
4C	42	138	69	5	130	436	442	1.38%	9.64
4D	42	138	69	9	75	556	571	2.70%	5.68
5A	35	130	65	3	220	423	423	0.00%	12.14
5B	35	130	65	4	165	451	451	0.00%	20.93
5C	35	130	65	5	130	480	**478**	**-0.42%**	16.91
5D	35	130	65	9	75	599	617	3.01%	52.73
6A	32	100	50	3	170	223	223	0.00%	6.03
6B	32	100	50	4	120	233	233	0.00%	44.18
6C	32	100	50	10	50	325	330	1.54%	37.99
7A	41	132	66	3	200	279	279	0.00%	70.68
7B	41	132	66	4	150	283	283	0.00%	7.26
7C	41	132	66	9	65	335	344	2.69%	34.63
8A	31	126	63	3	200	386	386	0.00%	1.28
8B	31	126	63	4	150	403	**401**	**-0.50%**	10.43
8C	31	126	63	9	65	544	585	7.54%	30.91
9A	51	184	92	3	235	323	332	2.79%	13.55
9B	51	184	92	4	175	331	332	0.30%	30.65
9C	51	184	92	5	140	339	344	1.47%	16.69
9D	51	184	92	10	70	413	426	3.15%	27.54
10A	51	194	97	3	250	428	431	0.70%	32.05
10B	51	194	97	4	190	436	443	1.61%	9.04
10C	51	194	97	5	150	451	462	2.44%	68.43
10D	51	194	97	10	75	552	565	2.36%	32.01
mean								1.40%	20.78

Table 2.2: Comparison on the Belenguer DCARP instances.

In view of the results from Tables 2.1 and 2.2, it is clear that the proposed GRASP algorithm is competitive in comparison with the three variants of the algorithm proposed by Maniezzo and Roffilli for the same sets of instances in [95].

We point out that in 3 out of the 56 problems, the proposed GRASP algorithm improves the best solutions found until the moment. These three solutions are the numbers marked in bold in Table 2.2. Furthermore, in 31 out of the 56 problems, the proposed GRASP obtains the same solutions, and finally, in 22 out of the 56 problems the algorithm is unable to find the best solutions, although in most cases this gap is small. The few problems that are presented occurred with the instances with the biggest number of routes, for which the GRASP is unable to find high-quality solutions.

Taking a look at the CPU times, we can conclude that the GRASP algorithm is very fast.

On the other hand, and as previously mentioned, we are interested in comparing our algorithm against three variants of the algorithm proposed by Maniezzo and Roffilli, specifically, MS, GA and DP. To this end, and in order to demonstrate the validity of our algorithm against similar algorithms, a statistical test is performed to show whether there is enough evidence to "reject" the proposed GRASP when it presents similar performance to the three algorithms proposed by Maniezzo and Rofilli: a multi-start algorithm; a genetic algorithm; and a data perturbation algorithm.

Table 2.3 shows the results obtained when the Kruskal-Wallis test

is applied (see [82]). As is well-known, the Kruskal-Wallis test is a non-parametric test to allow the comparison of more than two categorical, independent groups. In our case, the independent variable consists of four different algorithms and the dependent variable is the solution obtained. The null hypothesis of the Kruskal-Wallis test corresponds to the condition in which there are no differences between the results on applying the four algorithms. With the Kruskal-Wallis test, a chi-square statistic is used to evaluate differences in mean ranks to assess the null hypothesis that there is no difference between the results of the four algorithm. As can be observed in Table 2.3, this statistic is 0.051, and the degree of freedom, 3, represents the number of groups minus 1. The results of the analysis indicates that there is no significant difference between the four algorithms since the p-value is 0.997. Hence, it can be concluded that, on average, all four algorithms obtain similar results.

Chi-Square	0.051
Degree of Freedom	3
p-value	0.997

Table 2.3: Kruskal-Wallis Test.

2.4.2 Multi-Objective GRASP

Henceforth, we will deal with a bi-objective routing problem. In order to adapt the aforementioned single-objective GRASP, we will focus on the ideas proposed by Martí *et al.* (see [97]). They propose various

adaptations of a single-objective GRASP metaheuristic to solve multi-objective combinatorial optimization problems. In particular, Martí *et al.* describe several alternatives for the specialization of the construction and the local search phase of GRASP when two or more objectives are considered.

In the bi-objective GRASP, we will have two greedy functions: $g_1(c)$, which evaluates candidate element $c \in \mathcal{C}$ with respect to objective function f_1, which minimizes the total cost of all routes; and $g_2(c)$, which evaluates candidate element $c \in \mathcal{C}$ with respect to objective function f_2, which minimizes the cost of the route that has the maximum cost.

Let x be the partial solution under construction in a given iteration, and let $\mathcal{C}$ be the candidate set with all the remaining elements that can be added to x. We evaluate the elements in $\mathcal{C}$ with a function $g_i(c)$, for some $i = 1, 2$, thereby computing $RCL = \{c \in \mathcal{C} | g_i(c) \leq g_{i,min} + \alpha(g_{i,max} - g_{i,min})\}$, where $\alpha \in [0, 1]$ and $g_{i,min}$ and $g_{i,max}$ are the minimum and maximum evaluations of g_i in $\mathcal{C}$, respectively.

We distinguish between two types of strategies:

- **Pure**. In a pure construction, only one objective is considered during a single construction and it cannot be selected to construct again until the remaining objectives are also selected, i.e., in each construction a single different objective is considered.

- **Combined**. In combined construction, various objectives guide a single construction, i.e., in each construction all objectives will be considered.

In the local search, we use the same two main strategies as in the construction methods, pure and combined, according to the way in which the objective functions are selected.

Moreover, a new quality mechanism is also incorporated into the GRASP in order to save all the non-dominated solutions found.

It is worth mentioning that a major difference between a single-objective local search and a multi-objective local search is that in the former, we only need to check whether the final solution obtained (the local optima) improves upon the best known solution. In contrast, in a multi-objective local search, every solution visited has to be checked for its possible inclusion in the set of non-dominated solutions.

Construction Phase

Within the pure construction category, a **pure-ordered construction** will be implemented in which the objective to be considered in a construction is selected in an ordered fashion, whereby each construction is guided by a different objective. Specifically, in the first construction, the candidate elements are evaluated with $g_1(c)$, and in the second construction, with $g_2(c)$, and then we resort again to $g_1(c)$. In short, we follow the order of the objectives across the different constructions. In this way, each construction is expected to produce a solution of good quality with respect to the objective that is evaluated.

Within the combined construction category, a sequential combined method will be implemented in which each construction step is guided by

a different objective. Specifically, a greedy function g_i, for $i = 1, 2$, will be selected in an ordered fashion in each step of a given construction. This is known as an **ordered-sequential combined construction**. Hence, g_1 is used in step j, g_2 in step $j + 1$, and so on.

To sum up, two different construction methods will be implemented in the multi-objective GRASP: the **pure-ordered construction** and the **ordered-sequential combined construction**.

Once both bi-objective versions of the construction phase in the GRASP have been explained in broad terms, our new bi-objective construction phase of the GRASP for routing problems can be considered in greater detail.

Two construction algorithms are presented for each objective. The construction phase for the **first objective** (to minimize the total cost of all routes) coincides with the single-objective construction phase, i.e., the same two construction methods explained in Section 2.4.1. In each phase, nodes are assigned to routes according to the extra-cost function, $g^{(1)}$, that henceforth will be denoted as $g_1^{(1)}$, and according to the regret function, $g^{(2)}$, that from now on will be denoted as $g_1^{(2)}$. The terms $g_{min}^{(1)} = g_{1,min}^{(1)}$, $g_{max}^{(1)} = g_{1,max}^{(1)}$, $g_{min}^{(2)} = g_{1,min}^{(2)}$, and $g_{max}^{(2)} = g_{1,max}^{(2)}$ are also defined in a similar way. We have changed the notation to emphasize the fact that we are using the objective function, f_1. The GRASP approach constructs a RCL with "good" candidates, according to the greedy functions, $g_1^{(1)}$ and $g_1^{(2)}$. As it is customary in GRASP, an element of the RCL is selected at random and updates all the involved variables. This strategy adds

elements to routes while the capacity of a vehicle is not exceeded. The solution proposed is the best of the two obtained.

The construction phase for the **second objective** (to minimize the cost of the route that has the maximum cost) is defined in a similar way, i.e., as two construction algorithms. Two similar algorithms are designed to optimize the second objective function, f_2. In each phase, nodes are assigned to routes either according to the extra-cost function but only considering the route with maximum cost, $g_2^{(1)}$, or according to the regret function but fixing one of the routes to that route that has the maximum cost, $g_2^{(2)}$. Again, the solution proposed is the best of the two obtained.

The two constructive algorithms for the second objective are now explained in detail. Note that, in this case, the proposed construction algorithm is a sequential algorithm since one route at a time is built, the route that has the maximum cost.

- **Step 1** (compute a list of seeds). Choose K *seed nodes* to initialize each vehicle route, r_k, $k = 1, \ldots, K$. The seeds are chosen as those nodes hardest to visit. In an intermediate step, vertex $j^* \in V \backslash (S \cup \{0\})$ is chosen to be added to the set of seeds, S, from a set of candidates with greater values of the function $g^{(0)}(j) = \sum_{s \in S \cup \{0\}} \min\{d_G(s,j), d_G(j,s)\}$. No further details need be given because this function is the same as that in the single-objective GRASP algorithm. So, in the bi-objective algorithm, the seeds are chosen in the same way for the two objectives.

- **Step 2** (Assigning nodes to routes according to the extra-cost function by fixing the route that has maximum cost). Starting with the set of non-selected nodes, J, initially composed of all nodes except those of the depot and the seeds, and in an intermediate step, vertex $j^* \in J$ is chosen to be added to the route that has the maximum cost, which is denoted by k^*[6], at the position i in a greedy fashion from the subset of candidate elements, i.e., from the RCL, with lower values according the function $g_2^{(1)}(j, k^*, i) = c_{r_k^*(i-1)j} + c_{jr_k^*(i)} - c_{r_k^*(i-1)r_k^*(i)}$, where $r_k^*(i)$ denotes the i-th position of the route that has the maximum cost. Note that if the capacity of a vehicle is exceeded, then no further nodes can be allocated to this route. To build the RCL, we compute $g_{2,min}^{(1)} = \min_{\substack{j \in J \\ i=1,\ldots,|r_k^*|}} g_2^{(1)}(j, k^*, i)$ and $g_{2,max}^{(1)} = \max_{\substack{j \in J \\ i=1,\ldots,|r_k^*|}} g_2^{(1)}(j, k^*, i)$.

- **Step 3** (Assigning nodes to routes according to the regret function by fixing the route that has the maximum cost). Starting again with the set of non-selected nodes, J, in an intermediate step, vertex $j^* \in J$ is chosen to be added to route r_k^* at the position i from RCL with greater values according the function $g_2^{(2)}(j) = g_2^{(1)}(j, k^{**}, i_{k^{**}}) - g_2^{(1)}(j, k^*, i_{k^*})$, where k^* denotes the route that has the maximum cost, and k^{**} denotes the route with the second-lowest extra cost, i.e., $k^{**} = \arg \min_{k=1,\ldots,K\setminus\{k^*\}} g_2^{(1)}(j, k, i)$. Again if the capacity of a vehicle is exceeded, then no further nodes can be allocated to this

[6] k^* is not the same route for all the intermediate steps of the algorithm. To avoid cumbersome notation, we just consider k^*.

route. To build the RCL, we compute $g_{2,min}^{(2)} = \min_{j \in J} g_2^{(2)}(j)$ and $g_{2,max}^{(2)} = \max_{j \in J} g_2^{(2)}(j)$.

Note that the construction phase for the second objective is very similar to the construction phase for the first objective. The only difference is that the construction phase attending to the second objective is focused on the minimization of the cost of the route that has the maximum cost and not in all routes as happens in the construction phase for the first objective.

Having described the construction phase for both objectives, it is worth mentioning that they are included in two different versions of the bi-objective GRASP, the pure-ordered construction and the ordered-sequential combined construction; see Algorithm 4 and Algorithm 5 in detail.

Furthermore, a function is defined, $Insert\&Update(r)$, which inserts route r into the set of non-dominated solutions and removes those solutions in R dominated by r. Therefore, R keeps an archive of all non-dominated solutions found during the construction phase of the algorithms, and the local search phase then works on R and applies the local search to each non-dominated solution.

Local Search Phase

At this point, a feasible solution x is obtained from the constructive phase, either by using a pure constructive method or by using a combined constructive method. In the local search phase, the same two main

Algorithm 4 Multi-objective Construction Phase (pure-ordered)

Input: $G = (V, A)$, K, $\alpha^{(0)}$, $\alpha^{(1)}$, $\alpha^{(2)}$;
Output: $R = \{(r = (r_k)_{k=1,\dots,K})\}$ set of non-dominated solutions;
Initialize: $S = \{0\}$, $J = V \setminus \{0\}$, $k \leftarrow 1$, $R \leftarrow \emptyset$;
repeat
    For all $j \in J$ compute $g^{(0)}(j)$;
    Define $RCL^{(0)} \leftarrow \{j \in J | g^{(0)}(j) \geq g^{(0)}_{max} - \alpha^{(0)}(g^{(0)}_{max} - g^{(0)}_{min})\}$;
    Select j^* at random from $RCL^{(0)}(J)$;
    $r_k \leftarrow r_k \cup \{j^*\}$, $S \leftarrow S \cup \{j^*\}$, $J \leftarrow J \setminus \{j^*\}$, $k \leftarrow k + 1$;
until $k = K + 1$;
for all Objective $m = 1, 2$ **do**
    $r^{(1)} \leftarrow r$, $J^{(1)} \leftarrow J$;
    while $J^{(1)} \neq \emptyset$ **do**
        while the capacity of the K vehicles is not exceeded **do**
            For all $j \in J^{(1)}$ compute $g^{(1)}_m(j, k, i)$;
            Define $RCL^{(1)}_m \leftarrow \{j \in J^{(1)} | g^{(1)}_m(j, k, i) \leq g^{(1)}_{m,min} + \alpha^{(1)}(g^{(1)}_{m,max} - g^{(1)}_{m,min})\}$;
            Select j^* at random from $RCL^{(1)}_m(J^{(1)})$;
            $r^{(1)} \leftarrow r^{(1)} \cup \{j^*\}$, $J^{(1)} \leftarrow J^{(1)} \setminus \{j^*\}$;
        end while
    end while
    $R \leftarrow Insert\&Update(r^{(1)})$;
    $r^{(2)} \leftarrow r$, $J^{(2)} \leftarrow J$;
    while $J^{(2)} \neq \emptyset$ **do**
        while the capacity of the K vehicles is not exceeded **do**
            For all $j \in J^{(2)}$ compute $g^{(2)}_m(j)$;
            Define $RCL^{(2)}_m \leftarrow \{j \in J^{(2)} | g^{(2)}_m(j) \geq g^{(2)}_{m,max} - \alpha^{(2)}(g^{(2)}_{m,max} - g^{(2)}_{m,min})\}$;
            Select j^* at random from $RCL^{(2)}_m(J^{(2)})$;
            $r^{(2)} \leftarrow r^{(2)} \cup \{j^*\}$, $J^{(2)} \leftarrow J^{(2)} \setminus \{j^*\}$;
        end while
    end while
    $R \leftarrow Insert\&Update(r^{(2)})$;
end for
return R

Algorithm 5 Multi-objective Construction Phase (ordered-sequential combined)

Input: $G = (V, A)$, K, $\alpha^{(0)}$, $\alpha^{(1)}$, $\alpha^{(2)}$;
Output: $R = \{(r = (r_k)_{k=1,\dots,K})\}$ set of non-dominated solutions;
Initialize: $S = \{0\}$, $J = V \setminus \{0\}$, $k \leftarrow 1$, $R \leftarrow \emptyset$;
repeat
    For all $j \in J$ compute $g^{(0)}(j)$;
    Define $RCL^{(0)} \leftarrow \{j \in J | g^{(0)}(j) \geq g^{(0)}_{max} - \alpha^{(0)}(g^{(0)}_{max} - g^{(0)}_{min})\}$;
    Select j^* at random from $RCL^{(0)}(J)$;
    $r_k \leftarrow r_k \cup \{j^*\}$, $S \leftarrow S \cup \{j^*\}$, $J \leftarrow J \setminus \{j^*\}$, $k \leftarrow k + 1$;
until $k = K + 1$;
$r^{(1)} \leftarrow r$, $J^{(1)} \leftarrow J$;
while $J^{(1)} \neq \emptyset$ **do**
    while the capacity of the K vehicles is not exceeded **do**
        for all Objective $m = 1, 2$ **do**
            For all $j \in J^{(1)}$ compute $g^{(1)}_m(j, k, i)$;
            Define $RCL^{(1)}_m \leftarrow \{j \in J^{(1)} | g^{(1)}_m(j, k, i) \leq g^{(1)}_{m,min} + \alpha^{(1)}(g^{(1)}_{m,max} - g^{(1)}_{m,min})\}$;
            Select j^* at random from $RCL^{(1)}_m(J^{(1)})$;
            $r^{(1)} \leftarrow r^{(1)} \cup \{j^*\}$, $J^{(1)} \leftarrow J^{(1)} \setminus \{j^*\}$;
        end for
    end while
end while
$R \leftarrow Insert\&Update(r^{(1)})$;
$r^{(2)} \leftarrow r$, $J^{(2)} \leftarrow J$;
while $J^{(2)} \neq \emptyset$ **do**
    while the capacity of the K vehicles is not exceeded **do**
        for all Objective $m = 1, 2$ **do**
            For all $j \in J^{(2)}$ compute $g^{(2)}_m(j)$;
            Define $RCL^{(2)}_m \leftarrow \{j \in J^{(2)} | g^{(2)}_m(j) \geq g^{(2)}_{m,max} - \alpha^{(2)}(g^{(2)}_{m,max} - g^{(2)}_{m,min})\}$;
            Select j^* at random from $RCL^{(2)}_m(J^{(2)})$;
            $r^{(2)} \leftarrow r^{(2)} \cup \{j^*\}$, $J^{(2)} \leftarrow J^{(2)} \setminus \{j^*\}$;
        end for
    end while
end while
$R \leftarrow Insert\&Update(r^{(2)})$;
return R

strategies can be defined as in the construction phase. A local search can then be applied to the solution x using a **pure-ordered local search** or an **ordered-sequential combined local search**.

A **pure-ordered local search** attempts to improve a solution x guided by the objective function f_1, and ignores the remaining objectives. It should be borne in mind that the deterioration of the other objectives is permitted while f_1 is being improved. When objective f_1 cannot be further improved, it is then attempted to improve the solution by the objective f_2 until this cannot be further improved, and so on. The method halts when no objective can be improved further. The need to save the previous solution must be taken into consideration in order to prevent the occurrence of cycles.

An **ordered-sequential combined local search** attempts to improve a solution x by considering, at each step, an alternate objective function when selecting the best solution in the neighbourhood. In short, at step j, the solution is improved by considering the objective function f_1, at step $j+1$, the solution is improved by considering the objective function f_2, at step $j+2$ again by considering f_1, and so on. The method stops when no objective can be further improved. Note that the deterioration of other objectives is permitted.

As happens in the multi-objective construction phase, in the multi-objective local search phase, the neighbourhoods considered for the first objective, f_1, coincide with the neighbourhoods defined in the single-objective local search, see Section 2.4.1. These neighbourhood structures

are denoted by $\mathcal{N}_1$, $\mathcal{N}_2$, $\mathcal{N}_3$, $\mathcal{N}_4$ and $\mathcal{N}_5$.

It should be emphasized that new neighbourhood structures, are specifically designed to optimize the second objective, f_2. These are based on the previous classic neighbourhood structures adapted to the second objective. More precisely, the following neighbourhoods are described below:

1. **Neighbourhood structure 1 ($\mathcal{N}_1^*$):** The adaptation of the neighbourhood structure $\mathcal{N}_1$ by considering the objective function f_1 makes no sense when considering the objective function f_2. This neighbourhood structure cannot be adapted since we have no guarantee of reducing the cost of the route that has maximum cost by moving a set of nodes from different routes to another route.

2. **Neighbourhood structure 2 ($\mathcal{N}_2^*$):** It consists of swapping λ nodes from the route that has the maximum cost to another route, as long as this is feasible, where λ is the maximum number of nodes to be exchanged. This neighbourhood will be called *restricted* λ-*interchange*.

   In Figure 2.9 and 2.10, a restricted 2-interchange and a restricted 3-interchange are shown, respectively. Figure 2.9 shows the restricted 2-interchange, where two nodes are selected from the red route, that is the route that has the maximum cost, and are inserted in the blue route.

   Figure 2.10 shows the restricted 3-interchange, where three nodes

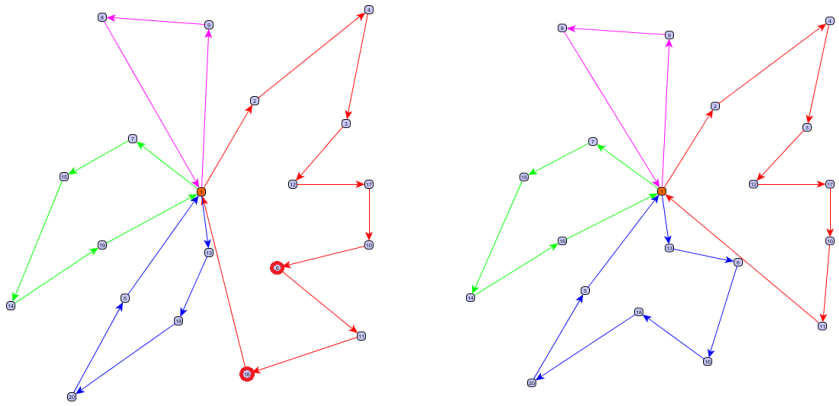


Figure 2.9: $\mathcal{N}_2^*$ with $\lambda = 2$

are selected from the red route, that is, the route with maximum cost, and are included in the pink route.

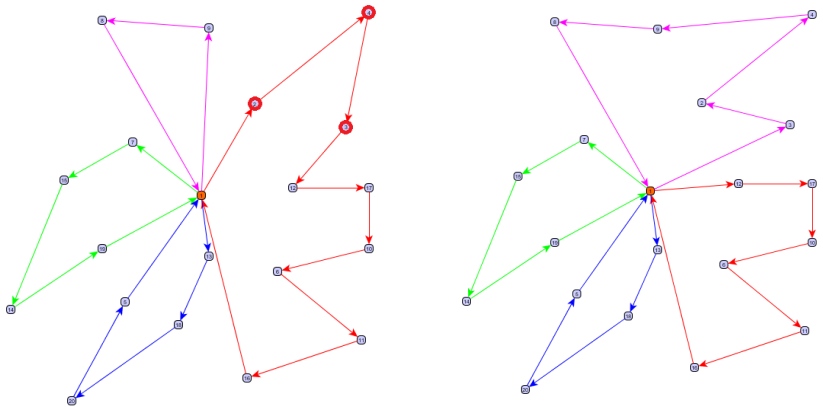


Figure 2.10: $\mathcal{N}_2^*$ with $\lambda = 3$

3. **Neighbourhood structure 3 ($\mathcal{N}_3^*$):** This movement evaluates the possibility of exchanging one node belonging to the route having maximum cost with a node of any other route, thus possibly reducing

the cost of the route that has the maximum cost, while maintaining feasibility. This neighbourhood will be called *balanced-exchange operator*.

Figure 2.11 shows the balanced-exchange operator, where one node is selected from the red route, that is, the route that has the maximum cost, another node is selected from the blue route, and these are exchanged.

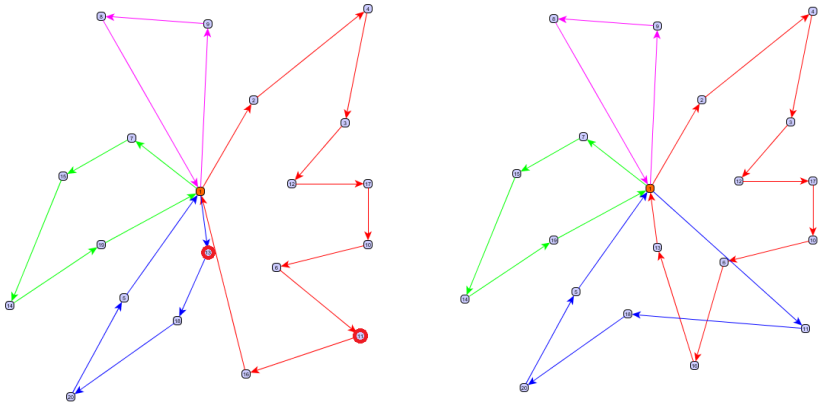


Figure 2.11: $\mathcal{N}_3^*$

4. **Neighbourhood structure 4 ($\mathcal{N}_4^*$):** This neighbourhood structure evaluates the possibility of moving a node belonging to the route that has the maximum cost to any other route in the best position. This movement is named as *balanced-relocate operator*.

   Figure 2.12 shows the balanced-relocate operator, where one node is selected from the red route, that is, the route that has the maximum cost, and is inserted into another route, specifically, the pink route.

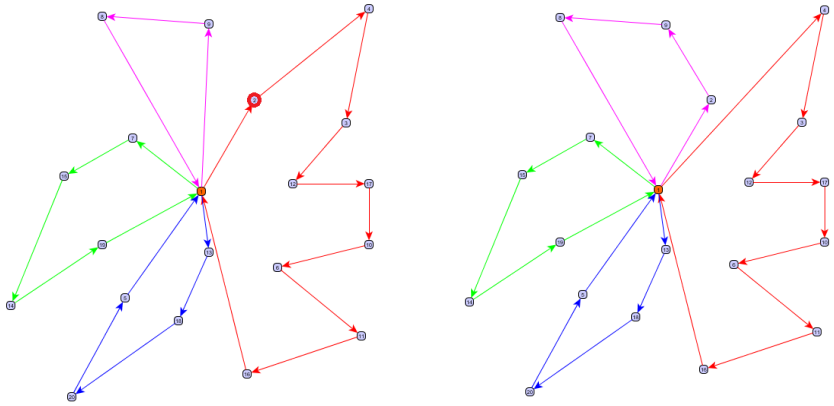


Figure 2.12: $\mathcal{N}_4^*$

5. **Neighbourhood structure 5 ($\mathcal{N}_5^*$):** Interchange two consecutive nodes within the route that has the maximum cost. This movement is performed when the route that has the maximum cost is reduced, see Figure 2.13.

   Figure 2.13 shows the $\mathcal{N}_5$ movement, where two consecutive nodes are selected from the red route, that is, the route that has the maximum cost, and they interchange their positions.

Pseudo-code for the pure-ordered local search phase of the bi-objective GRASP is given in Algorithm 6, while Algorithm 7 includes the pseudo-code for the ordered-sequential combined local search phase of the bi-objective GRASP.

To sum up, four algorithms are obtained:

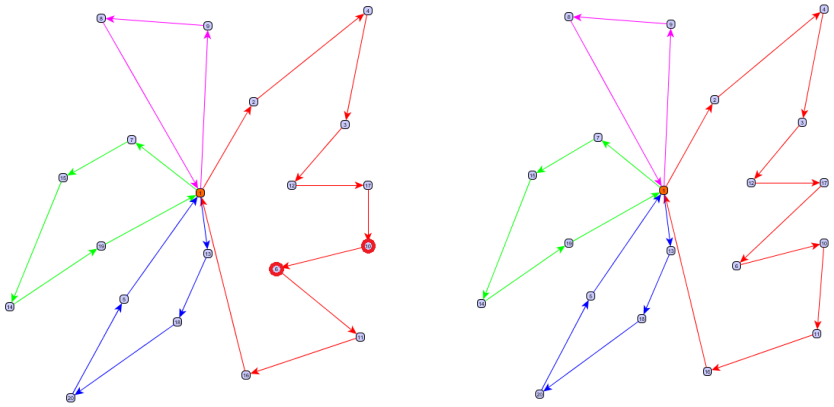


Figure 2.13: $\mathcal{N}_5^*$

- **Cpure LSpure:** Multi-objective GRASP performed by a pure-ordered construction and by a pure-ordered local search.

- **Ccomb LSpure:** Multi-objective GRASP performed by an ordered-sequential combined construction and by a pure-ordered local search.

- **Cpure LScomb:** Multi-objective GRASP performed by a pure-ordered construction and by an ordered-sequential combined local search.

- **Ccomb LScomb:** Multi-objective GRASP performed by an ordered-sequential combined construction and by an ordered-sequential combined local search.

Algorithm 6 Multi-objective Local Search Phase (VND pure-ordered)

Define the set of neighbourhood structures $\mathcal{N}_m$ and $\mathcal{N}_m^*$, $m = 2, \ldots, m_{max}$
Input: $R = \{(r = (r_k)_{k=1,\ldots,K})\}$ set of non-dominated solutions;
Initialize: $m \leftarrow 2$;
for all $r \in R$ **do**
    repeat
        local search: $r' \leftarrow arg\min \mathcal{N}_m(r)$;
        if $f_1(r') < f_1(r)$ **then**
            $R \leftarrow Insert\&Update(r')$ and $m \leftarrow 2$ (centre the search around r and search again in the first neighbourhood);
        else
            $m \leftarrow m + 1$ (switch to another neighbourhood);
        end if
    until $m = m_{max}$;
    repeat
        local search: $r' \leftarrow arg\min \mathcal{N}_m^*(r)$;
        if $f_2(r') < f_2(r)$ **then**
            $R \leftarrow Insert\&Update(r')$ and $m \leftarrow 2$ (centre the search around r and search again in the first neighbourhood);
        else
            $m \leftarrow m + 1$ (switch to another neighbourhood);
        end if
    until $m = m_{max}$;
end for
return R

Algorithm 7 Multi-objective Local Search Phase (VND ordered-sequential combined)

Define the set of neighbourhood structures $\mathcal{N}_m$ and $\mathcal{N}_m^*$, $m = 2, \ldots, m_{max}$
Input: $R = \{(r = (r_k)_{k=1,\ldots,K})\}$ set of non-dominated solutions;
for all $r \in R$ **do**
    Initialize: $m \leftarrow 2$;
    repeat
        local search: $r' \leftarrow arg\min \mathcal{N}_m(r)$;
        if $f_1(r') < f_1(r)$ **then**
            $R \leftarrow Insert\&Update(r')$;
            local search: $r' \leftarrow arg\min \mathcal{N}_m^*(r)$;
        else if $f_2(r') < f_2(r)$ **then**
            $R \leftarrow Insert\&Update(r')$;
        else
            $m \leftarrow m + 1$ (switch to another neighbourhood);
        end if
    until $m = m_{max}$;
end for
return R

Computational Results

In this section, the proposed algorithms on the two previous sets of benchmark instances from Maniezzo and Roffilli are tested. However, two objectives are considered, f_1 and f_2: to minimize the total cost of all routes and to minimize the cost of the route that has the maximum cost, respectively.

To the best of our knowledge, no algorithms have yet been designed to solve DCARPs or ACVRPs: only undirected versions have been found. Therefore our four algorithms will be compared against each other in order to illustrate the performance of the algorithms for routing problems on directed graphs.

Each problem is solved using our Cpure LSpure, Ccomb LSpure, Cpure LScomb, and Ccomb LScomb GRASP algorithm. The non-dominated or efficient solutions for each problem and for each algorithm are provided in Appendix A. Furthermore, whenever possible, the performance metrics explained in Section 2.3 will be given, i.e., the Coverage Metric, the Hypervolume Indicator, the Unary Epsilon Indicator, and the R_2 Indicator.

The same parameter settings are considered as stated in the single-objective GRASP algorithm. Each algorithm is run three times, considering $10,000$ iterations or 180 seconds as the maximum. Seeds are chosen at random, and the inclusion of the nodes in the construction phase selects the best nodes, instead of one at random from among the best candidates, that is, $\alpha^{(0)} \in [0, 1]$, $\alpha^{(1)} = \alpha^{(2)} = 0$.

Table 2.4 shows average values of the Epsilon Indicator, the Hypervolume and the R_2 Indicator for both sets of DCARP instances (56 problems). Table 2.5 presents the coverage between all pairs of these four algorithms under consideration. In Appendix A, the set of non-dominated solutions and their metrics are provided for each problem when more than one solution is obtained for each algorithm. Otherwise, only the set of non-dominated solutions is provided.

Clearly, Tables 2.4 and 2.5 show that the best overall algorithm is the Ccomb LScomb algorithm. As can be observed, the Ccomb LScomb algorithm provides better approximations to the efficient frontiers than the other strategies. The Epsilon Indicator, the Hypervolume and the R_2

Indicator support this point since the Ccomb LScomb algorithm presents lower values than the other algorithms. The coverage metric confirms this observation, with the following inequalities:

• $\mathcal{C}$(Ccomb LScomb, Cpure LSpure)> $\mathcal{C}$(Cpure LSpure, Ccomb LScomb),

• $\mathcal{C}$(Ccomb LScomb, Ccomb LSpure)> $\mathcal{C}$(Ccomb LSpure, Ccomb LScomb), and

• $\mathcal{C}$(Ccomb LScomb, Cpure LScomb)> $\mathcal{C}$(Cpure LScomb, Ccomb LScomb).

To conclude, it is worth mentioning that the average number of non-dominated solutions are 2.74, 2.88, 2.33, and 2.65 for the Cpure LSpure, the Ccomb LSpure, the Cpure LScomb, and the Ccomb LScomb GRASP algorithms, respectively.

	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.315	0.264	0.315	**0.075**
Hypervolume	0.322	0.624	0.204	**0.053**
R_2 Indicator	0.102	0.085	0.065	**0.013**

Table 2.4: Metrics: Epsilon Indicator, Hypervolume, and R2 Indicator

Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.252	0.116	**0.092**
Ccomb LSpure	0.407	-	0.254	**0.136**
Cpure LScomb	0.511	0.407	-	**0.160**
Ccomb LScomb	**0.636**	**0.563**	**0.456**	-

Table 2.5: Coverage metric

Chapter 3

Waste Collection Problem

3.1 Introduction

Waste management is one of the most difficult and expensive operational problems faced by local authorities or private enterprises in any large city. Waste management is made up of the collection, transport, processing, recycling, disposal, and monitoring of waste materials. Naturally, collection is the most important and costly aspect in the cycle. For an overview of most operational problems related to waste management planning, we refer to Golden *et al.* (see [64]), and Ghiani *et al.* (see [57]), whereas reviews of practical applications are given in Vigo *et al.* (see [130]).

As pointed out by Hemmelmayr *et al.*, see [71], the design and management of an effective collection service involves several steps, which in turn can be grouped into three main phases. The first phase is the "Demand Analysis", where through statistical models and specific data collection, the quantification of the waste to be collected is performed. The second phase is that of "Offer Planning", which involves the choice of the

specific collection model, namely door-to-door service (ARP) or based on large street bins (VRP), to be used for each waste fraction to be collected. In addition, the appropriate service frequency and day combinations to collection sites must be determined. Finally, the last phase is the "Design of the Vehicle Routes" for the collection vehicles on each day of a planning horizon. In this dissertation, we will focus on this last phase since the other phases are already stipulated by LIPASAM, the local company which manages the urban waste of Seville.

This chapter is structured as follows. The Waste Collection Problem (WCP) existing in Seville is described and formulated after some of the most recent related work about waste collection problem is briefly explained in Section 3.2. Section 3.3 then describes the study area, i.e., the geographical description of the city, the existing waste collection system in Seville, and the Geographical Information System used to obtain the data information in order to use it in the waste collection optimization. In Section 3.4, the problem is modelled as a Directed Capacitated Arc Routing Problem (DCARP) in which some variants are introduced and compared to those in previous related work. First, we consider two special nodes instead of one: the depot and the landfill. Furthermore, the fleet of vehicles is not necessarily identical, this variant is also presented in Chapter 1 as the Heterogeneous Fleet Routing Problem. It is also assumed that vehicles perform more than one trip (this variant is called Routing Problem with Multiple Trips). In addition, we have modelled the WCP in Seville as a DCARP because it is the most natural formulation for this problem. In

this section, the formulation is presented and some computational results are reported. Specifically, the validity of the formulation is tested on a number of small and medium-sized problems in the literature using exact algorithms with an optimization software package, CPLEX version 12.2. In addition, the limitations to finding optimal solutions using exact algorithms are presented. Finally, in order to solve these limitations, in Section 3.5, the single-objective GRASP presented in Chapter 2 is adapted to solve the WCP in Seville. Similarly, the best multi-objective GRASP from among the four variations of the algorithm are adapted to solve the WCP in Seville. Solutions obtained for the two approaches are presented and these routes are then compared against the current routes provided by the company.

3.2 Related Work

In the last two decades, local authorities have devoted increasing attention to waste collection mainly due to its impact on public concern about environment. This service is usually divided into three major areas: residential, commercial, and industrial. Each area includes municipal solid waste (and eventually, recycling) material, and each is very different from the others, i.e., number of routes, frequency of service, size of the container, etc. In general, this essential service involves great operational costs. In fact, efficient management of solid waste collection can generate significant savings while still ensuring hygiene standards and satisfaction

of the inhabitants.

Researchers try to reduce the costs by improving the routing of waste collection, and by finding the most suitable location of disposal facilities, the location of containers, as well as by minimizing the number of vehicles used. An early paper on waste collection was published in 1974 by Beltrami and Bodin, see [16]. They focused on waste collection activities in New York City (USA) and were interested in the design of routes in accordance with feasible combinations of days for the collection of containers for exactly a preset number of times. The objective was to service all containers assigned each day and to minimize the overall routing cost. In particular, they explored a variety of routing procedures and addressed the problem in two different ways, they clustered first and then optimized each of the routes or they routed first and then partitioned this giant tour into feasible routes.

Since then, numerous researchers have begun to study similar models and have developed methods to solve the WCP. As the evolution of the number of studies on waste increases over the years, in this review we will only mention the most recent work related to the waste collection problem.

In 2000, Bodin *et al.*, see [20], defined a sanitation routing problem called the Rollon-Rolloff Vehicle Routing Problem. In this problem, tractors move large trailers, in which the waste is collected, between locations and a disposal facility. One tractor can only transport one trailer at a time because of the huge size of the trailers. Bodin *et al.* defined the

problem and a mathematical programming formulation was presented. Furthermore, they developed two lower bounds and four heuristic algorithms which were tested on 20 different problems. These problems involved up to 199 trips and a single disposal facility.

In the same year, Mourão and Almeida, see [103], studied the WCP in a quarter of Lisbon (Portugal). They modelled it as a capacitated arc routing problem with side constraints. The problem consists of minimizing the collection cost of the waste, but allowing multiple trips. To this end, they developed two lower-bounding methods and a three-phase heuristic. The same problem was studied by Mourão and Amado a few years later, see [104], whereby was presented a heuristic method that provides good feasible solutions. Experimental results shown that the new proposal outperforms the previous model in terms of both quality and running time.

In 2007, Simonetto and Borenstein, see [125], tested a decision-support system over the WCP in Porto Alegre (Brazil). The authors stated that it is possible to obtain a mean reduction of 8.82% in the distance to be covered and a reduction of 17.89% in the weekly number of trips by the collection vehicles. This result is highly significant to the Municipal Department of Urban Cleaning because it can represent savings of around 10% of the annual budget for solid waste collection per year, in terms of operational and maintenance costs. One year later, Li *et al.*, see [91], designed a truck schedule operation plan for solid waste collection in the same city. They first solved a simplification of the original WCP,

without considering capacity constraints of the vehicles. Additionally, the solution was allowed to contain imbalanced trips. A second model was proposed in order to balance the trips. The authors proposed a heuristic approach to solve both models. Experimental results indicated that the new method significantly improved system performance in comparison to that of manual planning (savings of 25.24% for the number of vehicles and 27.21% for the average distance travelled).

In the same year, Bautista *et al.*, see [10], presented a methodology for solving the waste collection problem in Sant Boi de Llobregat, within the metropolitan area of Barcelona (Spain). The WCP was modelled as an arc routing problem and the graph was then transformed into a node routing problem including forbidding turns. The problem was solved through the ant colonies heuristic and various neighbourhood strategies were compared. The solutions obtained improved considerably on those that the city had been using. In that paper, the vehicle fleet was of a fixed number, and hence any reduction obtained was in the operating cost and the acoustic contamination. Those reductions are measured by means of fuel consumption, total time that the trucks are collecting waste, and length of the routes.

Finally, Maniezzo and Roffilli, see [95], worked on the WCP by considering 9 instances of two towns in Italy. The size of the largest instances were up to $1,400$ containers. The authors modelled the WCP as a directed capacitated arc routing problem. They proposed a modified

Iterated Variable Neighbourhood Search implementing three different shaking strategies, the first is a random multi-start heuristic, the second is a genetic algorithm, and the third is a data perturbation procedure. Experimental results concluded that this last method obtains the best results. Note that all methods start by constructing a solution with a two-phase constructive procedure based on the ideas presented in Christofides *et al.*, see [28].

On the other hand, if we focus on the multi-objective version of waste collection problem, to the best of our knowledge, only one paper has addressed two objectives. In 2006, Lacomme *et al.*, see [85], modelled the WCP as an undirected CARP in which the total cost and the makespan were simultaneously minimized. In order to solve the problem, a NSGA-II algorithm was implemented using good heuristics (Path-Scanning, Augment-Merge and Ulusoy's heuristics) in the initial population, and a local search capable of improving the solutions for both criteria were added. The algorithm was originally designed to be applied to the town of Troyes (France), however, insufficient data availability cause this algorithm to be tested on a set of benchmark instances.

3.3 Description of the Study Area

The geographic location of our study is described in terms of population, waste generation rate, waste density in containers, number and location of containers for each fraction, and the characteristics of the streets (i.e.,

length, average collection, vehicle speed, street width, traffic direction, and vehicle access). Data was provided by the local authorities, regional studies on municipal waste, official statistics, and the company responsible for the disposal of Seville's waste.

3.3.1 Geographic Description

This study is focused on Seville, a city of southern Spain (see Figure 3.1 to gain an impression of the geographical disposition of this city). Seville is the capital city of the autonomous community of Andalusia and of the province of Seville. It is situated on the plain of the Guadalquivir River (the longest river in Andalusia), with an average elevation of 7 metres above sea level. The population of this capital city was 703, 021 (according to official statistics of 2011) and it is ranked as the fourth largest city of Spain.

Seville has a hot-summer mediterranean climate with semi-arid climate influences. The annual average temperature is 18.6 C (65 F). Winters are mild, January is the coolest month and summers are blazing hot; July is the warmest month. Precipitation varies from 600 to 800 mm (millimeters) per year, concentrated in the period from October to April. December is the wettest month.

Figure 3.1: Province of Seville where Seville city is located in the southwest

3.3.2 Existing System of Waste Collection

LIPASAM holds the municipal cleaning duties of Seville City Council: responsible for the cleaning of 1,077 Km of roads, the municipal waste collection, and waste treatment to save resources and prevent environmental pollution.

According to the data information provided by LIPASAM as of 2012, the budget allocation to this company amounts to 92.6 million euros of which 63.3 million euros constitute staff costs. Furthermore, the workforce is made up of 1,492 workers. More information can be found

on the company's website (www.lipasam.es).

On the other hand and according to Spanish legislation, there are various types of municipal solid waste that are collected separately, and as a result, specific containers for each waste type:

i) organic matter (brown or grey containers)

ii) glass (green containers),

iii) paper and cardboard (blue containers), and

iv) light weight packaging (yellow containers).

Our goal is to improve the collection and transportation of the organic matter.

The solid waste collection, specifically that of organic matter, is carried out in a mechanized was throughout the city, 365 days a year. It is the local company which manages the urban waste of Seville, LIPASAM, which has fixed the appropriate service frequency of seven days per week.

On the other hand, the type of vehicle used for waste collection depends on the type of container as well as the characteristics of the city streets. The streets can be classified as wide or narrow, streets in the city are divided according to the type of container and every container is associated to a type of vehicle: side load (side-load compactor) or rear load (rear-load compactor, mini compactor, bimodal compactor and mini-bimodal compactor). Figure 3.2 shows a side-load and a rear-load compactor.

Figure 3.2: Side load and rear load

In this work, the focus is placed on the side-load collection since LIPASAM is interested in reducing the number of rear-load containers and increasing the number of side-load containers, as far as possible. This is the consequence of financial stringency: side-load containers have a higher capacity than rear-load containers. Increasing the size of the containers implies a reduction in the number of containers located on the streets and this, in turn, implies a reduction of the routing cost over the time. Both the formulation and the GRASP algorithm, presented in the following sections, could easily be applied to any kind of container by just modifying the specification.

In the case of side-load collection, no distinction is needed between wide and narrow streets, because for side-load compactors, only wide streets are considered. Specifically, $2,821$ containers must be collected, 128 containers with a capacity of $2,400$ litres and $2,693$ containers with a capacity of $3,200$ litres and moreover, a fleet of 32 vehicles with a capacity of between $10,500$ and $11,500$ kg each is available. The number of vehicles in the fleet available in the company is much greater than the number of

vehicles needed to collect the waste in the city: nowadays, only 20 vehicles are in use. Assuming that each vehicle can collect 65 containers, it is worth simplifying the problem by transforming the capacity of vehicles into a discrete variable. Henceforth, the demand is measured as the number of containers that a vehicle can collect.

Reliable information on the mass and volume of waste in each container is not available. The amount of municipal solid waste is highly variable and the accumulation of waste depends on several factors, such as the number of inhabitants using a container, GDP per capita, lifestyle, time of the year, etc. Therefore, the WCP considered is stochastic by nature. In this study, the average accumulation rate of waste in each container is estimated statistically using historical data. The time to unload a vehicle at the waste disposal facility as well as the time to empty each waste container is also estimated based on historical data.

3.3.3 Geographical Information System

The first phase of our study required the use of Geographical Information System (GIS) tools to build a spatial data base with detailed digital information in the form of geographic objects, container locations, and street characteristics. To this end, the ArcGIS 9.3 software platform has been applied.

ArcGIS is a commercial Geographical Information System that has been widely used in many areas where spatially-enabled data needs to be stored, retrieved, analyzed, visualized and even served online. We have

used ArcGIS in two ways: as a platform to store all problem data in geographic format and to visualize them; and as the solutions we obtain through our approximate approach.

In order to work in a GIS environment, several input data layers are stored in ArcGISs File Geodatabase format as points and lines.

- Road network. A network is a system of interconnected elements. It can be visualized as a series of lines connecting points in a city, where the lines represent fragments of streets connected to each other at intersections, and points represent the intersection of at least two fragments of streets.

- Depot and Landfill location. These are two special points in the network associated with a line, and represent the place where vehicles are stored, and the place where vehicles unload the waste, respectively.

- Container locations. These are points in the network associated with a line with an assigned weight and represents the demand of each street. That means, if a street has several containers placed on it, then they are aggregated at only one point whose demand is the sum of the number of containers.

- Street directions. There are two possibilities: one way or both ways. One-way streets are represented with an arc where the origin and destination are coherent with the direction of the street. Similarly,

two-way streets are represented with two different arcs, each arc representing one of the two possible directions.

- Turn restrictions. Forbidden turns imply major constraints for the definition of feasible routes. These limitations are imposed not only due to traffic regulations, but also when the roads are narrow, since the size of the trucks make it impossible to make a turn. All limitations have been considered in the GIS.

- Speed limits. Speed limits are imposed based on the maximum speed allowed and by the traffic volume of each street.

The network road or street map of Seville is depicted in Figure 3.3. The blue point represents the depot, the red point represents the landfill, and yellow points represent containers. Finally, grey lines represents streets, and hence the set of all grey lines represents the road network. This map results in a directed graph with $99,429$ vertices (street crossings and dead-end streets) and $224,638$ arcs (fragment of streets). The map of the city contains $49,669$ turns, out of which $16,752$ are forbidden. The vast majority of forbidden turns are U-turns. Required arcs, $1,642$ arcs, are associated with roads where containers are placed. Containers are located on the pavements next to the roads, and a total of $2,821$ containers must be collected, according to data provided by the company as of February 2012. Furthermore, at that moment, there are 20 routes to perform the collection of the containers and the average total distance travelled every day is approximately $1,440$ km.

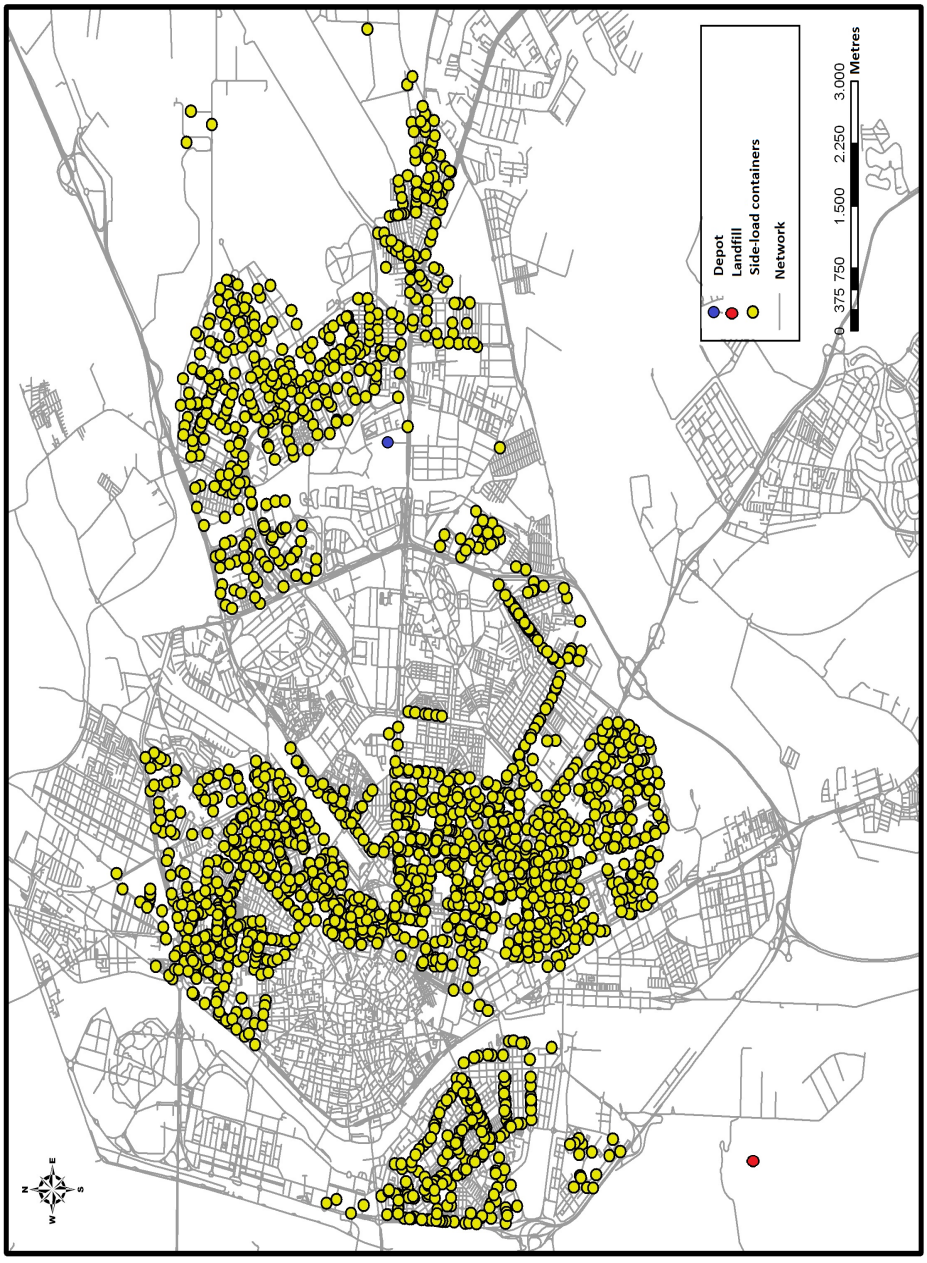


Figure 3.3: Geographical disposition of side-load containers

3.4 Waste Collection Problem. An exact approximation

As previously mentioned in the first chapter, a routing problem may be classified as a Vehicle Routing Problem (VRP) or as an Arc Routing Problem (ARP), depending on where the demand occurs on the underlying road network. Specifically, in the collection of normal residual waste, citizens usually deposit their waste in special bags or containers in front of, or near to, their houses. Hence, if the collection is performed along the streets, then the problem can be modelled as an ARP, but at the same time, if the locations where the citizens deposit their waste is considered as a collection of points, then the WCP can be modelled as a VRP. In our case, the WCP will be modelled as an ARP not only because it is the more natural formulation for waste collection problems, see [39], but also because the data is specified in terms of streets. Note that if the data were specified in terms of points, the node routing problem approach would be adopted.

In this section, the WCP is formulated and then the problem is solved using exact algorithms for small and medium-sized problems. The formulation contains two different objective functions, although they are solved as separate problems. Due to the limitations of exact algorithms, in Section 3.5, approximate algorithms are used for solving our WCP in Seville: specifically, we will adapt the GRASP algorithm presented in Chapter 2 for both the single-objective problem and the multi-objective

problem.

3.4.1 A Formulation for the Waste Collection Problem

In this section, we model the WCP in Seville, whilst taking into account all the necessary issues to find an optimal or near-optimal solution.

The problem under consideration is formulated as a DCARP by generalizing the model of Gouveia *et al.* (see [66]) and introducing several new concepts: (i) two special nodes are introduced, the depot and the landfill also known as disposal facility (this situation can easily be extended to more than two special nodes); (ii) we formulate the concept of indexing the variables by routes and trips, because every route has different trips; (iii) additional constraints are included to ensure that a route depends on the type of trip (open or closed trip); (iv) a CARP with multiple trips is formulated since it is necessary to assign each vehicle to several trips during a working day. This situation arises since the capacity of vehicles is relatively small and the working time of each vehicle must be satisfied; and (v) we assume the vehicles may be of different types.

A detailed description of the WCP in Seville and the notation necessary for this formulation are now given. A weighted directed graph $G = (V, A)$ is considered. According to the information given by LIPASAM, there are two special nodes, called depot (where vehicles are stored) and landfill (where vehicles unload). Each vehicle must visit streets until its capacity is full and until it finishes the working day. Therefore each vehicle must perform a *route* that will be defined as a sequence of two

or more *trips*. In the initial trip or open trip, vehicles leave the depot and start to collect refuse and, once their capacity is full, they then go to the landfill. When vehicles empty the refuse into the landfill, they begin as many closed trips as necessary without exceeding the length of the working day. In these closed trips, vehicles start in the landfill and back to it when their capacity are full again. In closing, they perform a final return trip from the landfill to the depot. This trip is fixed. In fact, all vehicles perform at least two trips since the capacity of vehicles is relatively small and the working day must be satisfied. By a *vehicle's route*, we mean the total course driven per day since departing from the depot, until returning back to the depot. Figure 3.4 shows an example with only one route. Specifically, the first trip of the vehicle (depicted in red) departs from the depot, collects containers (without exceeding the maximum capacity) and dumps the collected waste at the landfill. It then performs one additional trip (depicted in blue), and finally, the vehicle ends the route by returning to the depot (depicted in green).

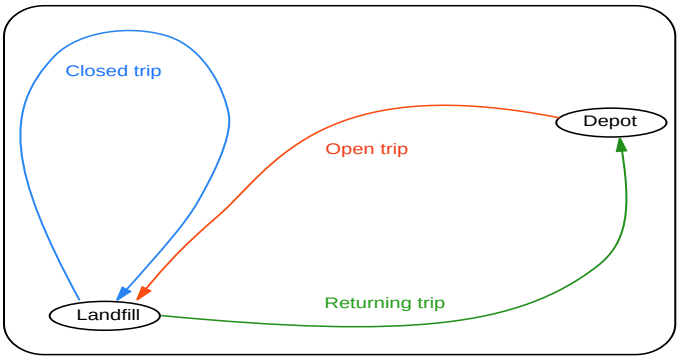


Figure 3.4: A typical route composed by two trips

The above trips have a cost regarding time or distance consumed along each traversed street. Specifically, streets with containers placed along them must be served because they have positive demands. These streets incur a cost that will be called *service cost*. Certain streets do not have containers placed along them and are traversed only to ensure the connectivity of the trips. Every street traversed by a vehicle without being served, incurs a cost that will be called *deadheading cost*. Simultaneously, each street, with or without demand, has another associated cost called *travel cost*.

Furthermore, we consider two fixed costs: a *fixed cost*, associated to each vehicle, that represents the time to travel from the landfill to the depot at the end of the working day, and a *dumping cost*, that is the time spent by a vehicle when it is dumping at the landfill.

It is important to note that service costs, deadheading costs, dumping cost, and the fixed cost are all measured in units of time, while travel costs are measured in units of length. This situation that has been presented is the most realistic and requires two weight or cost matrices associated with the graph (distance and time), however most authors unfortunately simplify the problem by considering only one of the cost matrices. In this dissertation both cost matrices will be considered without any simplification being made.

Consider the following notation:

- $G = (V, A)$ is the weighted directed graph where V is the set of nodes and A is the set of arcs.

- $0 \in V$ is the depot node where vehicles are stored.
- $l \in V$ is the landfill where vehicles are emptied.
- $\{1, 2, \ldots, n\} \in V$ corresponds to the nodes (street crossings and dead-end streets).
- $|V| = n + 2$ is the total number of nodes.
- $(i, j) \in A$ is the fragment of the street delimited by nodes from i to j and it is coherent with its direction.
- $|A|$ is the total number of arcs.
- $A_R \subseteq A$ is the set of required arcs in G and similarly, $|A_R|$ is the total number of required arcs.
- K denotes the number of vehicles.
- Q_k denotes the capacity of the vehicle k. Note that when $Q_k = Q$ for all k the fleet of vehicles is homogeneous.
- T is the maximum number of trips allowed on each route.
- t_{ij} is the deadheading cost (in units of time) of arc $(i, j) \in A$, i.e., the cost of traversing $(i, j) \in A$ without demand.
- s_{ij} is the service cost (in units of time) of arc $(i, j) \in A_R$, i.e., the cost of traversing $(i, j) \in A_R$ with demand.
- c_{ij} is the travel cost (in units of length) of arc $(i, j) \in A$, i.e., the cost of traversing $(i, j) \in A$.

- q_{ij} is the demand of arc $(i,j) \in A_R$, i.e., the demand to be served at $(i,j) \in A_R$.

- λ is the dumping cost, paid every time a vehicle is emptied at the landfill. Specifically, this cost is measured as the time that a vehicle needs to dump the waste in the landfill.

- μ is a fixed cost associated to each vehicle and represents the time to travel from the landfill to the depot at the end of the working day.

- W is the maximum working cost, i.e., the hours in a working day.

Once the notation has been introduced, the problem formulation as well as the decision variables can be presented.

- $x_{ij}^{tk} = \begin{cases} 1 & \text{if } (i,j) \in A_R \text{ is served by vehicle k in trip t} \\ 0 & \text{otherwise} \end{cases}$

- y_{ij}^{tk} is the number of times that arc $(i,j) \in A$ is deadheaded (i.e., traversed but not served) by vehicle k in trip t.

The objective functions considered for the WCP are: the total travel cost of all routes, see 3.4.1, (in the real-world problem, this is the total distance travelled by all vehicles measured in units of length); and the cost of the route with maximum cost, see 3.4.2, in order to balance the routes (in the real-world problem, this is the longest route measured in units of time). Both objective functions are of minimization. In our problem, another objective that could be considered, is the minimization of the number of vehicles. This objective takes discrete values and many

authors consider it as an input of the problem. We have also followed this last strategy, by considering the number of vehicles as an input of the problem and solving the problem for several desired values.

$$\min \sum_{k=1}^{K} \sum_{t=1}^{T} \left(\sum_{(i,j)\in A} c_{ij} y_{ij}^{tk} + \sum_{(i,j)\in A_R} c_{ij} x_{ij}^{tk} \right) \tag{3.4.1}$$

$$\min \max_{k=1,\ldots,K} \left[\sum_{t=1}^{T} \left(\sum_{(i,j)\in A} t_{ij} y_{ij}^{tk} + \sum_{(i,j)\in A_R} s_{ij} x_{ij}^{tk} \right) \right] \tag{3.4.2}$$
$$+\lambda \left[\sum_{t=1}^{T} \left(\sum_{(i,l)\in A} y_{il}^{tk} + \sum_{(i,l)\in A_R} x_{il}^{tk} \right) \right] + \mu \left(\sum_{(0,j)\in A} y_{0j}^{1k} + \sum_{(0,j)\in A_R} x_{0j}^{1k} \right)$$

All constraints considered for the WCP are given below and are explained step by step.

Conditions (3.4.3) and (3.4.4) impose the continuity of trips at each node. That is, the number of times that a vehicle arrives to an intermediate node must coincide with the number of times that a vehicle leaves this node. The first trips, $t = 1$, must be considered as different constraints because these are open trips, in contrast to the rest of the trips, $t = 2, \ldots, T$, which are closed trips. That is why we distinguish between constraints (3.4.3) and (3.4.4).

$$\sum_{j:(i,j)\in A} y_{ij}^{1k} + \sum_{j:(i,j)\in A_R} x_{ij}^{1k} = \sum_{j:(j,i)\in A} y_{ji}^{1k} + \sum_{j:(j,i)\in A_R} x_{ji}^{1k}, \tag{3.4.3}$$
$$i = 1, \ldots, n, \quad i \neq 0, l, \quad k = 1, \ldots, K$$

$$\sum_{j:(i,j)\in A} y_{ij}^{tk} + \sum_{j:(i,j)\in A_R} x_{ij}^{tk} = \sum_{j:(j,i)\in A} y_{ji}^{tk} + \sum_{j:(j,i)\in A_R} x_{ji}^{tk}, \tag{3.4.4}$$
$$i = 0, 1, \ldots, n, l, \quad k = 1, \ldots, K, \quad t = 2, \ldots, T$$

Conditions (3.4.5) mean the first trips should begin at the depot and finish at the landfill; (3.4.6) hold that if a vehicle begins a first trip then it must begin a second one, that is, only two trips are required, and (3.4.7) indicate that the other trips (third trip, fourth trip, and so on) are not required, but that they are only activated when needed.

$$\sum_{j:(0,j)\in A} y_{0j}^{1k} + \sum_{j:(0,j)\in A_R} x_{0j}^{1k} = \sum_{j:(j,l)\in A} y_{jl}^{1k} + \sum_{j:(j,l)\in A_R} x_{jl}^{1k}, \qquad (3.4.5)$$
$$k = 1, \ldots, K$$

$$\sum_{j:(0,j)\in A} y_{0j}^{1k} + \sum_{j:(0,j)\in A_R} x_{0j}^{1k} = \sum_{j:(l,j)\in A} y_{lj}^{2k} + \sum_{j:(l,j)\in A_R} x_{lj}^{2k}, \qquad (3.4.6)$$
$$k = 1, \ldots, K$$

$$\sum_{j:(l,j)\in A} y_{lj}^{(t+1)k} + \sum_{j:(l,j)\in A_R} x_{lj}^{(t+1)k} \leq \sum_{j:(l,j)\in A} y_{lj}^{tk} + \sum_{j:(l,j)\in A_R} x_{lj}^{tk} \qquad (3.4.7)$$
$$k = 1, \ldots, K, \quad t = 2, \ldots, T-1$$

The collection of refuse is guaranteed by (3.4.8); (3.4.9) impose the maximum working cost for each vehicle; (3.4.10) and (3.4.11) imply that the dumping cost and the fixed cost, λ and μ, are adequately charged in the objective function.

$$\sum_{k=1}^{K}\sum_{t=1}^{T} x_{ij}^{tk} = 1, \quad \text{for all } (i,j) \in A_R \qquad (3.4.8)$$

$$\mu + \sum_{t=1}^{T}\left(\sum_{(i,j)\in A} t_{ij} y_{ij}^{tk} + \sum_{(i,j)\in A_R} s_{ij} x_{ij}^{tk}\right) + \qquad (3.4.9)$$
$$\lambda \sum_{t=1}^{T}\left(\sum_{(i,l)\in A} y_{il}^{tk} + \sum_{(i,l)\in A_R} x_{il}^{tk}\right) \leq W, \quad k = 1, \ldots, K$$

$$\sum_{j:(0,j)\in A} y_{0j}^{1k} + \sum_{j:(0,j)\in A_R} x_{0j}^{1k} \leq 1, \quad k = 1, \ldots, K \tag{3.4.10}$$

$$\sum_{j:(l,j)\in A} y_{lj}^{tk} + \sum_{j:(l,j)\in A_R} x_{lj}^{tk} \leq 1, \quad k = 1, \ldots, K, \tag{3.4.11}$$
$$t = 2, \ldots, K$$

New decision variables, called flow variables, are now introduced. Flows, along a trip, are decreasing and show the order in which arcs must be traversed. Therefore, for each vehicle k, each trip t, and each arc:

- f_{ij}^{tk} is the flow in arc $(i, j) \in A$, related with the remaining demand by vehicle k in trip t.

Conditions (3.4.12), (3.4.13), (3.4.14), (3.4.15), and (3.4.16) are flow conservation constraints which, together with the linking constraints (3.4.17), force the connectivity in the trips and remove subtours. Note that (3.4.12) and (3.4.13) are typical generalized flow conservation constraints on each node i, guaranteeing that if arc (i, j) is served by vehicle k, then q_{ji} units of flow are absorbed by node i. Conditions (3.4.17) impose upper bounds on flow variables needed to guarantee the capacity constraints for each vehicle in each trip.

$$\sum_{j:(j,i)\in A} f_{ji}^{1k} - \sum_{j:(i,j)\in A} f_{ij}^{1k} = \sum_{j:(j,i)\in A_R} q_{ji} x_{ji}^{1k}, \tag{3.4.12}$$
$$k = 1, \ldots, K, \quad i = 1, \ldots, n, \quad i \neq 0, l$$

$$\sum_{j:(j,i)\in A} f_{ji}^{tk} - \sum_{j:(i,j)\in A} f_{ij}^{tk} = \sum_{j:(j,i)\in A_R} q_{ji} x_{ji}^{tk}, \tag{3.4.13}$$
$$k = 1, \ldots, K, \quad t = 2, \ldots, K, \quad i = 0, 1, \ldots, n, \quad i \neq l$$

$$\sum_{j:(0,j)\in A} f_{0j}^{1k} = \sum_{(i,j)\in A_R} q_{ij} x_{ij}^{1k}, \quad k = 1, \ldots, K \tag{3.4.14}$$

$$\sum_{j:(l,j)\in A} f_{lj}^{tk} = \sum_{(i,j)\in A_R} q_{ij} x_{ij}^{tk}, \tag{3.4.15}$$
$$k = 1, \ldots, K, \quad t = 2, \ldots, T$$

$$\sum_{i:(i,l)\in A} f_{il}^{tk} = \sum_{i:(i,l)\in A_R} q_{il} x_{il}^{tk}, \tag{3.4.16}$$
$$k = 1, \ldots, K, \quad t = 1, \ldots, T$$

$$f_{ij}^{tk} \leq Q_k (y_{ij}^{tk} + x_{ij}^{tk}), \quad \text{for all } (i,j) \in A, \tag{3.4.17}$$
$$k = 1, \ldots, K, \quad t = 1, \ldots, T$$

Finally, (3.4.18), (3.4.19) and (3.4.20) define the nature of the decision variables.

$$x_{ij}^{tk} \in \{0, 1\}, \quad \text{for all } (i,j) \in A_R, \tag{3.4.18}$$
$$k = 1, \ldots, K, \quad t = 1, \ldots, T$$

$$y_{ij}^{tk} \geq 0, \text{ integer for all } (i,j) \in A, \tag{3.4.19}$$
$$k = 1, \ldots, K, \quad t = 1, \ldots, T$$

$$f_{ij}^{tk} \geq 0, \quad \text{for all } (i,j) \in A, \tag{3.4.20}$$
$$k = 1, \ldots, K, \quad t = 1, \ldots, T$$

Once the problem has been formulated, it will be proved the validity of the model for the Capacitated Arc Routing Problem. The following lemmas are needed to prove it. Note that the proofs are simple adaptations of those in [66].

Lemma 3.4.1. *The vehicle capacity requirements are satisfied by any feasible solution of the problem.*

Proof.

Case 1. If $t = 1$ and $i = 0$, and (3.4.17) holds for all arcs $(0, j) \in A$, then we obtain $\sum_{j:(0,j)\in A} f_{0j}^{1k} \leq Q_k \cdot \sum_{j:(0,j)\in A}(y_{0j}^{1k} + x_{0j}^{1k})$, for $k = 1, \ldots, K$. Using (3.4.10), the previous inequality implies $\sum_{j:(0,j)\in A} f_{0j}^{1k} \leq Q_k$, for $k = 1, \ldots, K$.

Finally, (3.4.14) gives a required inequality $\sum_{(i,j)\in A_R} q_{ij} x_{ij}^{1k} \leq Q_k$, for $k = 1, \ldots, K$, which completes the proof for the open trip, $t = 1$.

Case 2. If $t \neq 1$ and $i = l$, and (3.4.17) holds for all arcs $(l, j) \in A$, then we obtain $\sum_{j:(l,j)\in A} f_{lj}^{tk} \leq Q_k \cdot \sum_{j:(l,j)\in A}(y_{lj}^{tk} + x_{lj}^{tk})$, for $k = 1, \ldots, K$. Using (3.4.11), the previous inequality implies $\sum_{j:(l,j)\in A} f_{lj}^{tk} \leq Q_k$, for $k = 1, \ldots, K$.

Finally, (3.4.15) gives a required inequality $\sum_{(i,j)\in A_R} q_{ij} x_{ij}^{tk} \leq Q_k$, for $k = 1, \ldots, K$, which completes the proof for the closed trips, $t \neq 1$. □

The following lemma shows that any feasible solution of the problem is connected, that is, subtours cannot be obtained.

Lemma 3.4.2. *For each vehicle $k = 1, \ldots, K$ and each trip $t = 1, \ldots, T$, the graph induced by the set of arcs corresponding to the variables $x_{ij}^{tk} = 1$ and $y_{ij}^{tk} > 0$ is connected.*

Proof.

Case 1. If $t = 1$, then we prove this result by showing that any vehicle that serves required arcs in a set $S \subset V \setminus \{0, l\}$ starts at the depot and finishes at the landfill.

Assume that there is a set of nodes, S, served by vehicle k with S representing a connected component of required arcs, and that $0 \in \bar{S} = V \setminus S$ and $l \in \bar{S} = V \setminus S$.

From (3.4.12), we obtain

$$\sum_{i \in S} \left(\sum_{j:(j,i) \in A} f_{ji}^{1k} - \sum_{j:(i,j) \in A} f_{ij}^{1k} \right) = \sum_{i \in S} \left(\sum_{j:(j,i) \in A_R} q_{ji} x_{ji}^{1k} \right) =$$

$$= Q_S^{1k} + Q_{\bar{S},S}^{1k},$$

where Q_S^{1k} is the demand served by vehicle k in the first trip in subset S and $Q_{\bar{S},S}^{1k}$ is the demand served by vehicle k in the first trip in arcs from $\bar{S}$ to S.

By defining

$$f_{X,Y}^{1k} = \sum_{(i,j) \in (X,Y): X,Y \subset V} f_{ij}^{1k}$$

then the expression

$$\sum_{i \in S} \left(\sum_{j:(j,i) \in A} f_{ji}^{1k} - \sum_{j:(i,j) \in A} f_{ij}^{1k} \right)$$

can be rewritten as $f_{S,S}^{1k} + f_{\bar{S},S}^{1k} - f_{S,S}^{1k} - f_{S,\bar{S}}^{1k}$, leading to $f_{\bar{S},S}^{1k} = f_{S,\bar{S}}^{1k} + Q_S^{1k} + Q_{\bar{S},S}^{1k}$.

Since $f_{S,\bar{S}}^{1k} \geq 0$; $Q_{\bar{S},S}^{1k} \geq 0$ and, by assumption, $Q_S^{1k} > 0$ must hold, we obtain $f_{\bar{S},S}^{1k} > 0$. Using (3.4.17), we conclude that $y_{\bar{S},S}^{1k} > 0$ or

$x^{1k}_{\bar{S},S} > 0$. Therefore, the vehicle k in the first trip links $\bar{S}$ to S. Furthermore, by (3.4.5), the vehicle k in the first trip links S to $\bar{S}$, and hence, the vehicle k in the first trip starts at the depot and finishes at the landfill.

Case 2. If $t \neq 1$, then we prove this result by showing that any vehicle that serves required arcs in a set $S \subset V \setminus \{l\}$ starts and finishes at the landfill.

Assume that there is a set of nodes S, served by a vehicle k with S representing a connected component of required arcs, and that $l \in \bar{S} = V \setminus S$.

From (3.4.13), we obtain

$$\sum_{i \in S} \left(\sum_{j:(j,i) \in A} f^{tk}_{ji} - \sum_{j:(i,j) \in A} f^{tk}_{ij} \right) = \sum_{i \in S} \left(\sum_{j:(j,i) \in A_R} q_{ji} x^{tk}_{ji} \right) =$$

$$= Q^{tk}_{S} + Q^{tk}_{\bar{S},S},$$

where Q^{tk}_{S} is the demand served by vehicle k in a trip t in subset S, and $Q^{tk}_{\bar{S},S}$ is the demand served by vehicle k in the trip t in arcs from $\bar{S}$ to S.

By defining

$$f^{tk}_{X,Y} = \sum_{(i,j) \in (X,Y): X,Y \subset V} f^{tk}_{ij}$$

then the expression

$$\sum_{i \in S} \left(\sum_{j:(j,i) \in A} f^{tk}_{ji} - \sum_{j:(i,j) \in A} f^{tk}_{ij} \right)$$

can be rewritten as $f^{tk}_{S,S} + f^{tk}_{\bar{S},S} - f^{tk}_{S,S} - f^{tk}_{S,\bar{S}}$, leading to $f^{tk}_{\bar{S},S} = f^{tk}_{S,\bar{S}} + Q^{tk}_{S} + Q^{tk}_{\bar{S},S}$.

Since $f^{tk}_{S,\bar{S}} \geq 0$; $Q^{tk}_{\bar{S},S} \geq 0$ and that, by assumption, $Q^{tk}_{S} \geq 0$ must hold, then we obtain $f^{tk}_{\bar{S},S} > 0$. Using (3.4.17), we conclude that $y^{tk}_{\bar{S},S} > 0$ or $x^{tk}_{\bar{S},S} > 0$. Therefore, the vehicle k in the trip t links $\bar{S}$ to S, and is linked to the landfill. □

The two above lemmas and conditions (3.4.3), (3.4.4), (3.4.6) and (3.4.7) show that a feasible solution for the previous formulation can be represented as a set of trips for each vehicle. Together with constraint (3.4.8), the solution also satisfies all services. Thus, it now becomes trivial to prove Proposition 3.4.3.

Proposition 3.4.3. *The formulation is valid for the CARP.*

3.4.2 Computational Results

In this section, the validity of the formulation is tested on a set of benchmark instances. Computational experiments were performed on an hp Pavilion dm4 (Intel $CORE^{TM}$ i5 - 430M processor 2.26 GHz and 2GB RAM) with CPLEX 12.2.

The formulation contains two separate objective functions: f_1 (see equation 3.4.1) which represents the total travel cost; and f_2 (see equation 3.4.2) which represents the cost of the route with maximum time cost. Thus, we have solved each problem twice, the first time using f_1 and the second time using f_2.

Once again, two sets of instances will be considered in this section in order to illustrate the limitations of exact methods. As there is no set of instances in the literature with the structure of our problem, we have taken the DCARP instances from Maniezzo and Rofilli, see [95], adapted to the proposed formulation, by doubling the depot node (to represent the depot and the landfill) and connecting these identical nodes with zero-cost arcs. To this end, in Tables 3.1 and 3.2, one extra node is added onto the second column and the number of arcs and required arcs are computed once this extra node has been added. Furthermore, in these instances, only one cost is associated with each arc of the graph (the travel cost) and hence the constraint on the maximum hours in a working day is not considered.

Additionally, all problems were solved taking $T = 3$, that is, each route has three trips as maximum, one open and two closed. The dumping cost and the fixed cost (or travelling cost from the landfill to the depot at the end of the working day) have always been considered zero-cost, $\lambda = 0$ and $\mu = 0$.

The column headings are as follows:

- $|V|$ is the total number of nodes.
- $|A|$ is the number of arcs.
- $|A_R|$ is the number of required arcs.
- K the maximum number of vehicles.
- Q the maximum capacity of vehicles.

- f_1 represents the total cost of all routes.
- f_2 represents the cost of the route that has the maximum cost.
- CPU time represents the computing time in seconds ("tle", time limit exceeded, indicates that the half-an-hour limit was attained and CPLEX was halted).

The computational results presented in Tables 3.1 and 3.2 show the payoff matrix. The elements of the payoff matrix are obtained by optimizing each of the objectives f_1 and f_2 individually and then calculating the value of the other objective using the solution vector of the decision variables. Therefore, the diagonal elements of the payoff matrix are the optimum values for each individual goal, also known as "Ideal Point" or extreme values.

In view of these results, in most instances the computational time taken to solve the problem considering objective f_2 is greater than that of objective f_1. Note that when "tle" is obtained, then the optimal solution has not been found, just a feasible solution. Naturally, when only one vehicle is used, then the solution vector obtained by optimizing f_1 coincides with the solution vector obtained by optimizing f_2.

In the light of the results of Tables 3.1 and 3.2, we can conclude that even when we are solving small and medium-sized problems, CPLEX is unable to find optimal solutions for all problems in a reasonable amount of time. Hence, our proposed GRASP, as presented in Chapter 2, will be adapted in order to address the real-world problem.

It would be possible to examine and analyze the results of the following tables in greater detail. First, f_1 and f_2 values differ when $k > 1$, thereby demonstrating that these two objectives come into conflict. Second, f_1 and f_2 values should allow us to measure the effectiveness of the GRASP proposed in Chapter 2, since the same extreme values should be obtained. In the case of f_1, the GRASP found those values. However, in the case of f_2, it remains impossible to verify since different problems are being solved in the sense that the GRASP solved the problem considering that each vehicle only performs one trip and the exact formulation considers that each vehicle performs a maximum of three trips. Finally, it is worth mentioning that with a "large" number of arcs and number of vehicles, about $|A| > 55$ and $|K| > 2$, CPLEX remains unable to find an optimal solution. In relation with the real-world problem, it stands to reason that the problem is impossible to solve using exact algorithms.

Problem	$\|V\|$	$\|A\|$	$\|A_R\|$	K	Q		f_1	f_2	CPU
DA1	12	54	22	2	5	f_1	316	160	656.33
						f_2	316	158	195.17
DA2	12	66	26	2	5	f_1	339	200	673.66
						f_2	341	171	876.80
DA3	12	58	22	2	5	f_1	275	163	8.89
						f_2	275	138	99.89
DA4	11	50	19	2	5	f_1	287	152	5.48
						f_2	287	144	30.03
DA5	13	64	26	2	5	f_1	377	213	111.06
						f_2	377	191	683.77
DA6	12	56	22	2	5	f_1	298	216	20.31
						f_2	298	150	30.42
DA7	12	58	22	2	5	f_1	325	204	216.19
						f_2	325	163	848.58
DA10	27	60	37	4	27	f_1	394	133	tle
						f_2	426	108	tle
DA11	27	108	51	4	27	f_1	341	120	tle
						f_2	347	87	tle
DA12	12	58	25	2	10	f_1	304	172	8.84
						f_2	304	152	9.74
DA14	13	56	23	4	35	f_1	458	240	tle
						f_2	468	160	tle
DA15	10	66	28	2	41	f_1	550	362	tle
						f_2	564	283	tle
DA16	7	56	21	2	21	f_1	100	60	2.09
						f_2	100	50	16.74
DA17	7	56	21	2	37	f_1	58	43	tle
						f_2	58	29	618.31
DA18	8	72	28	2	24	f_1	127	77	1.67
						f_2	127	29	tle
DA19	8	72	28	2	41	f_1	91	66	0.39
						f_2	91	46	tle
DA20	9	90	36	2	37	f_1	170	93	0.30
						f_2	170	82	5.62
DA21	10	32	11	1	27	f_1	55	55	0.03
						f_2	55	55	0.02
DA22	10	58	22	2	27	f_1	121	62	1.17
						f_2	121	61	81.78
DA23	11	86	33	2	27	f_1	156	96	0.69
						f_2	156	78	7.16
DA24	11	110	44	3	27	f_1	200	80	145.67
						f_2	204	68	tle
DA25	12	132	55	4	27	f_1	235	78	tle
						f_2	241	61	tle

Table 3.1: DeArmon DCARP instances.

Problem	$\|V\|$	$\|A\|$	$\|A_R\|$	K	Q		f_1	f_2	CPU
1A	24	90	39	1	200	f_1	173	173	0.09
						f_2	173	173	0.13
1B	24	90	39	1	120	f_1	179	179	1.79
						f_2	179	179	1.87
1C	24	90	39	3	200	f_1	266	113	tle
						f_2	266	90	tle
2A	24	74	34	1	180	f_1	227	227	0.11
						f_2	227	227	0.13
2B	24	74	34	1	120	f_1	259	259	7.55
						f_2	259	259	7.63
2C	24	74	34	3	40	f_1	467	190	tle
						f_2	507	170	tle
3A	24	76	35	1	80	f_1	81	81	0.05
						f_2	81	81	0.06
3B	24	76	35	1	50	f_1	87	87	2.65
						f_2	87	87	2.53
3C	24	76	35	3	20	f_1	142	68	tle
						f_2	150	50	tle
4A	41	144	69	1	225	f_1	400	400	0.67
						f_2	400	400	0.58
4B	41	144	69	2	140	f_1	442	256	tle
						f_2	448	226	tle
4C	41	144	69	2	130	f_1	428	245	tle
						f_2	428	215	tle
5A	34	136	65	1	220	f_1	423	423	0.58
						f_2	423	423	0.73
5B	34	136	65	2	165	f_1	449	226	72.54
						f_2	449	225	tle
5C	34	136	65	2	130	f_1	474	284	tle
						f_2	478	239	tle
6A	31	110	50	1	170	f_1	223	223	0.37
						f_2	223	223	0.33
6B	31	110	50	2	120	f_1	233	176	87.70
						f_2	233	117	1267.15
6C	31	110	50	4	50	f_1	344	117	tle
						f_2	347	88	tle
7A	40	150	66	1	200	f_1	279	279	1.97
						f_2	279	279	1.97
7B	40	150	66	2	150	f_1	283	164	20.12
						f_2	283	142	tle
7C	40	150	66	3	65	f_1	341	125	tle
						f_2	342	115	tle
8A	30	136	63	1	200	f_1	386	386	0.41
						f_2	386	386	0.37
8B	30	136	63	2	150	f_1	395	208	46.63
						f_2	395	198	tle
8C	30	136	63	3	65	f_1	730	256	tle
						f_2	714	239	tle
9A	50	196	92	1	235	f_1	323	323	0.44
						f_2	323	323	0.41
9B	50	196	92	2	175	f_1	326	170	100.92
						f_2	326	163	242.77
9C	50	196	92	2	140	f_1	332	204	47.05
						f_2	332	166	1022.51
10A	50	202	97	1	250	f_1	428	428	0.66
						f_2	428	428	0.56
10B	50	202	97	2	190	f_1	436	221	tle
						f_2	436	218	tle
10C	50	202	97	2	150	f_1	454	286	tle
						f_2	466	233	tle

Table 3.2: Belenguer DCARP instances.

3.5 The Real-World Waste Collection Problem

At this point, we are ready to solve the real-world WCP in Seville. In short, we have described and formulated the problem, but as mentioned before, due to the size of the problem, approximate algorithms are needed for it to be solved. Hence, in this section, two algorithms, as presented in Chapter 2, are used and several adaptations are introduced. The two algorithms used are the single-objective GRASP, and one of the four multi-objective GRASPs introduced in Chapter 2.

It is well-known that most of the real problems defined on road networks are modelled on quite sparse graphs. We take advantage of working with a sparse and directed graph by transforming the DCARP graph into an Asymmetric Vehicle Routing Problem (ACVRP), whereby each required arc (i.e., each arc with positive demand) is replaced with one node. Note that the depot and the landfill are maintained because they are represented by two special nodes.

In mathematical terms, the original graph $G(V, A)$ is transformed into a new graph $\hat{G} = (\hat{V}, \hat{A})$, as explained in Chapter 1. Each arc $(i, j) \in A_R$ is then substituted by a vertex, m_{ij}, into the original arc and this new vertex is associated to a demand $q_{ij} > 0$. Therefore, $\hat{V} = \bigcup_{(i,j)\in A_R}\{m_{ij}\}\bigcup\{0, l\}$. By considering that the new graph is complete, it trivially holds that $\hat{A} = \hat{V} \times \hat{V}$.

Given two vertices, $m_{ij}, m_{kl} \in \hat{V}$ in the new graph, costs can then be computed by considering, on the one hand, the service and deadheading

costs (in units of time) and, on the other hand, the travel cost (in units of length). The new costs are then defined as follows:

$$\hat{t}_{m_{ij}m_{kl}} = s_{ij} + d_{G(t)}(j,k)$$

$$\hat{c}_{m_{ij}m_{kl}} = c_{ij} + d_{G(c)}(j,k)$$

where $d_{G(t)}(j,k)$ and $d_{G(c)}(j,k)$ are the shortest paths between vertices $j \in V$ and $k \in V$ in the graph G, using the deadheading costs, and the travel costs, respectively. In order to compute the new matrices, the Dijkstra's algo-rithm was implemented.

This transformation proves itself to be highly useful since the number of required arcs $|A_R| = 1,642$ is small, relative to the total number of arcs $|A| = 224,638$. Furthermore, the original graph, G, has $|V| = 99,429$ nodes and the transformed graph, $\hat{G}$, has $|\hat{V}| = 1,644$ nodes including those of the depot and the landfill. This considerable reduction implies fewer variables and, as a consequence, a shorter computing time to solve the problem. Only one single previous computation must be performed: the computation of the shortest paths.

Therefore, at this point the graph has been transformed into an ACVRP, and is associated to two types of weights or costs.

3.5.1 Single-Objective Waste Collection Problem

In this section, the single-objective GRASP is adapted to solve the single-objective WCP in Seville, whose objective is to minimize the total cost of all routes. In our case, this entails the minimization of the distance

travelled by all vehicles to collect the waste, while imposing a time constraint on each route. To this end, only the construction phase needs a slight modification: two special nodes are considered, open and closed trips are built, and each route has a maximum duration. The local search phase remains unchanged; only two constraints need to be borne in mind, the capacity and the working day, instead of only the capacity as in Chapter 2.

One of the main difficulties of the WCP in Seville is the huge instance that we have to deal with. At first, when we designed the constructive algorithm, the fact that the algorithm must be suitable for this kind of instance was taken into account. Hence, two algorithms are used to construct the initial routes. The first algorithm of the construction is based on the extra cost, and the second algorithm uses the regret heuristic. The returned solution is the best between of the two solutions. Algorithm 8 shows the pseudo-code of the constructive procedure. As input parameters, the algorithm receives the graph, $\hat{G}$, the number of vehicles, K, and the parameters for the RCLs, $\alpha^{(0)}$, $\alpha^{(1)}$ and $\alpha^{(2)}$ (line 1), and returns a feasible set of routes (line 2). Both algorithms start by initializing the set of routes with the highest cost, that is, the streets whose containers are the hardest to be served (lines 4 to 9).

First, in order to select the hardest streets to be served, we consider the greedy function, $g^{(0)}$, which computes the distance between streets with unserved containers, j, and streets with served containers, s (plus depot and landfill). In mathematical terms, if we have the set of streets

with served containers, S, the greedy function is computed as $g^{(0)}(j) = \sum_{s \in S \cup \{0,l\}} \min\{\hat{c}_{sj}, \hat{c}_{js}\}$ (line 5). A pure greedy strategy would select the element with the maximum value of $g^{(0)}$. Instead the GRASP approach constructs a RCL (line 6) with "good" candidates (according to the greedy function). As it is customary in GRASP, an element of the RCL is selected at random (line 7) and all the involved variables are updated (line 8). The process is maintained (lines 5 to 10) until the K routes are initialized with only one element.

This strategy can be adapted to initialize only closed trips when the open trips have been built. In particular, instead of setting S with the depot and the landfill (line 3), we initialize it only with the landfill (line 18 and 28) and redirect the execution of the code to the line 4.

The first construction algorithm is also based on the GRASP methodology (lines 10 to 19). In this case, the greedy function, $g^{(1)}$, is the *extra-cost function*. It computes the change in the objective function when inserting each street with unserved containers j at the best position i in route k, i.e., it computes the extra mileage. In mathematical terms, $g^{(1)}(j,k,i) = \hat{c}_{r_k(i-1)j} + \hat{c}_{jr_k(i)} - \hat{c}_{r_k(i-1)r_k(i)}$, where $r_k(i)$ denotes the vertex in the i-th position on route r_k (line 13). Therefore, as is customary in a GRASP design, the algorithm constructs the RCL (line 14), selects an element at random (line 15), and updates the affected variables (line 16). This strategy adds elements to routes, while the capacity of a vehicle or the working-time constraint is not exceeded.

The second construction algorithm (lines 20 to 29) starts from the

same partial solution as the first construction algorithm (see line 20) and considers the *regret function* as the greedy function, $g^{(2)}$. It computes the difference in distance when inserting each street with unserved containers j into the best route and when inserting it into the second-best route (obviously in their best position). More formally, $g^{(2)}(j) = g^{(1)}(j, k^2, i_{k^2}) - g^{(1)}(j, k^1, i_{k^1})$, where k^1 denotes the route with the lowest extra mileage, k^2 denotes the route with the second-lowest extra mileage, and i_k denotes the best position of route k (line 23). In mathematical terms, we compute $k^1 = \arg \min_{k=1,\ldots,K} g^{(1)}(j, k, i)$ and $k^2 = \arg \min_{k=1,\ldots,K \setminus \{k^1\}} g^{(1)}(j, k, i)$. The remaining steps are addressed in the same way as in the first construction algorithm.

The proposed GRASP construction does not guarantee the feasibility of the obtained solution (although these details have not been introduced into the pseudo-code for the sake of simplicity). When this happens, the solution is discarded and the procedure starts a new iteration to construct a new solution. Only feasible solutions are submitted to the local search phase. Fixing and improving unfeasible solutions implies extremely long computational time, which in our case, can run into hours.

As previously stated, the local search phase remains unchanged and can be used for our WCP. The only consideration is that every movement requires two constraints to be checked: the capacity of each vehicle and the duration of each route.

Algorithm 8 Construction Phase

1: **Input:** $\hat{G} = (\hat{V}, \hat{A})$, K, $\alpha^{(0)}$, $\alpha^{(1)}$, $\alpha^{(2)}$;
2: **Output:** $r = (r_k)_{k=1,\dots,K}$ feasible set of routes;
3: **Initialize:** $S = \{0, l\}$, $J = \hat{V} \setminus \{0, l\}$, $k \leftarrow 1$, $r \leftarrow \emptyset$, $f^* \leftarrow \infty$;
4: **repeat**
5: For all $j \in J$ compute $g^{(0)}(j)$;
6: Define $RCL^{(0)} \leftarrow \{j \in J | g^{(0)}(j) \geq g^{(0)}_{max} - \alpha^{(0)}(g^{(0)}_{max} - g^{(0)}_{min})\}$;
7: Select j^* at random from $RCL^{(0)}(J)$;
8: $r_k \leftarrow r_k \cup \{j^*\}$, $S \leftarrow S \cup \{j^*\}$, $J \leftarrow J \setminus \{j^*\}$, $k \leftarrow k + 1$;
9: **until** $k = K + 1$;
10: $r^{(1)} \leftarrow r$, $J^{(1)} \leftarrow J$;
11: **while** $J^{(1)} \neq \emptyset$ **do**
12: **while** the capacity or the working day of the K vehicles is not exceeded **do**
13: For all $j \in J^{(1)}$ compute $g^{(1)}(j, k, i)$;
14: Define $RCL^{(1)} \leftarrow \{j \in J^{(1)} | g^{(1)}(j, k, i) \leq g^{(1)}_{min} + \alpha^{(1)}(g^{(1)}_{max} - g^{(1)}_{min})\}$;
15: Select j^* at random from $RCL^{(1)}(J^{(1)})$;
16: $r^{(1)}_k \leftarrow r^{(1)}_k \cup \{j^*\}$, $J^{(1)} \leftarrow J^{(1)} \setminus \{j^*\}$;
17: **end while**
18: Go to line 4, $S = \{l\}$, $J \leftarrow J^{(1)}$, $k \leftarrow 1$;
19: **end while**
20: $r^{(2)} \leftarrow r$, $J^{(2)} \leftarrow J$;
21: **while** $J^{(2)} \neq \emptyset$ **do**
22: **while** the capacity or the working day of the K vehicles is not exceeded **do**
23: For all $j \in J^{(2)}$ compute $g^{(2)}(j)$;
24: Define $RCL^{(2)} \leftarrow \{j \in J^{(2)} | g^{(2)}(j) \geq g^{(2)}_{max} - \alpha^{(2)}(g^{(2)}_{max} - g^{(2)}_{min})\}$;
25: Select j^* at random from $RCL^{(2)}(J^{(2)})$;
26: $r^{(2)}_k \leftarrow r^{(2)}_k \cup \{j^*\}$, $J^{(2)} \leftarrow J^{(2)} \setminus \{j^*\}$;
27: **end while**
28: Go to line 4, $S = \{l\}$, $J \leftarrow J^{(2)}$, $k \leftarrow 1$;
29: **end while**
30: **if** $f(r^{(1)}) < f(r^{(2)})$ **then**
31: $r \leftarrow r^{(1)}$; $f^* \leftarrow f(r^{(1)})$;
32: **else**
33: $r \leftarrow r^{(2)}$; $f^* \leftarrow f(r^{(2)})$;
34: **end if**
35: **return** r

Computational Results

A description is now given of the results obtained when applying the methodology described below to the WCP in Seville, while taking into account just one objective, the minimization of the total distance travelled by all vehicles, i.e., the number of kilometers travelled by all vehicles.

In order to provide a variety of solutions to the company, the real-world problem is solved while taking three different values of the number of vehicles. Of course, the first value for the number of vehicles is the lowest value for which a feasible solution is found.

Three different problems are solved considering 19, 20 and 21 vehicles. The proposed GRASP is run ten times for each problem and as consequence 30 solutions are obtained. The maximum CPU time is set at $3,000$ seconds, so in total the computer runs for a maximum of $90,000$ seconds. Furthermore, the seeds are chosen in each run, taking $\alpha^{(0)}$ at random from $[0,1]$, $\alpha^{(1)} = 0.01$, and $\alpha^{(2)} = 0.01$.

Table 3.3 shows some statistic values to summarize the solutions obtained: the maximum, the minimum, and the mean distance travelled by all vehicles. Furthermore, the range and the mean range are provided, i.e, the difference between the maximum and the minimum, and the division of this difference by the number of vehicles, respectively. Finally, we show the standard deviation and Pearson's coefficient of variation in order to illustrate the level of homogeneity of the routes.

Figure 3.5 shows the trade-off between the total distance and the

	19 routes	20 routes	21 routes
Maximum	1410.56 km	1379.90 km	1333.74 km
Minimum	1313.34 km	1311.22 km	1276.51 km
Mean	1363.91 km	1339.19 km	1310.06 km
Range	97.22 km	68.68 km	57.23 km
Mean Range	5.12 km	3.43 km	2.73 km
Standard Deviation	34.80 km	20.74 km	18.03 km
CV	0.03	0.02	0.01

Table 3.3: Summary of routes

number of vehicles, i.e., how the total distance changes according to the number of routes. The x-axis represents the number of vehicles and the y-axis represents the total number of kilometers travelled by all vehicles. Results were obtained from the ten solutions, one for each run. This figure reports the maximum, the minimum and the mean for the total kilometres travelled for each number of vehicles. It can be observed that the greater the number of vehicles, the lower the total of kilometres travelled.

In view of the results of Tables 3.3 and Figure 3.5, we can affirm that the greater the number of routes, the shorter the total distance travelled by all vehicles. The best solutions could be considered to be those using 21 vehicles, however each vehicle also has an associated fixed cost, an economic cost. We therefore left the company to carry out the selection of the most suitable solution, according to their preferences.

If we focus on the economic cost, large savings can be achieved because the company uses 20 vehicles to collect the waste in Seville and we are able to obtain a solution using 19 vehicles. That is, the fixed cost

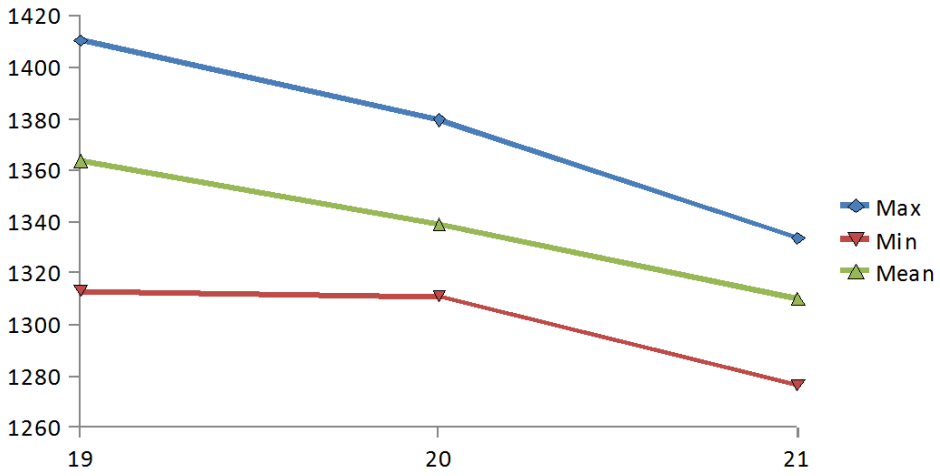


Figure 3.5: Total kilometres and number of routes

of one vehicle can be removed, which implies also removing the salary of one employee. On the other hand, if we take a look at the row *Minimum* from Table 3.3 (since these are the solutions we provide to the company), the difference between solutions using 19 and 20 vehicles is less than 3 kilometres and the difference between solutions using 19 and 21 vehicles is nearly 37 kilometres every day. In this situation, the company needs to seriously consider whehter it makes sense to pay the salary of one or two employees in order to save 3 or 37 kilometres every day, respectively.

The best out of the ten solutions for each number of vehicles is reported. Specifically, Tables 3.4, 3.5, and 3.6 show information on the planning of the service using 19, 20, and 21 routes, respectively. Column *Time* gives the time (in hours) that each vehicle route needs to perform its working day; column *Demand* counts the number of containers that

each vehicle collects; column *Req.Arcs* computes the number of required arcs; and column *Trips* counts the number of times that each vehicle goes to the landfill to unload.

See Appendix A to have a visual impression of the three solutions provide to the company (LIPASAM).

	Time	Demand	Req.Arcs	Trips
Route 1	6.00	156	90	3
Route 2	6.88	155	95	3
Route 3	6.59	156	99	3
Route 4	6.32	156	89	3
Route 5	6.54	156	91	3
Route 6	6.47	155	99	3
Route 7	6.47	155	105	3
Route 8	6.83	140	72	3
Route 9	6.78	156	92	3
Route 10	6.99	153	92	3
Route 11	6.60	156	82	3
Route 12	6.84	156	83	3
Route 13	6.42	130	73	2
Route 14	6.54	145	77	3
Route 15	6.95	154	76	3
Route 16	6.94	153	87	3
Route 17	5.44	129	78	2
Route 18	6.22	130	86	2
Route 19	5.33	130	76	2
	123.14	2821	1642	53

Table 3.4: 19 routes with a total cost of 1313.34 km

	Time	Demand	Req.Arcs	Trips
Route 1	5.91	150	97	3
Route 2	6.44	150	86	3
Route 3	5.95	149	87	3
Route 4	6.01	147	92	3
Route 5	5.98	129	73	2
Route 6	6.85	150	88	3
Route 7	6.32	148	89	3
Route 8	6.69	149	90	3
Route 9	5.51	129	73	2
Route 10	5.30	130	69	2
Route 11	6.86	150	87	3
Route 12	6.97	148	87	3
Route 13	6.99	149	92	3
Route 14	6.80	146	73	3
Route 15	6.94	147	80	3
Route 16	5.60	130	74	2
Route 17	5.06	130	70	2
Route 18	5.85	130	73	2
Route 19	4.99	130	78	2
Route 20	5.67	130	84	2
	122.67	2821	1642	52

Table 3.5: 20 routes with a total cost of 1311.22 km

	Time	Demand	Req.Arcs	Trips
Route 1	6.32	142	76	3
Route 2	5.47	142	92	3
Route 3	6.22	142	87	3
Route 4	6.43	142	93	3
Route 5	6.16	142	83	3
Route 6	6.18	142	81	3
Route 7	6.78	141	86	3
Route 8	5.57	130	81	2
Route 9	6.67	138	79	3
Route 10	4.55	130	79	2
Route 11	5.12	130	80	2
Route 12	5.07	130	70	2
Route 13	4.73	130	68	2
Route 14	4.86	130	79	2
Route 15	5.41	130	74	2
Route 16	5.74	130	71	2
Route 17	6.36	130	77	2
Route 18	5.51	130	72	2
Route 19	5.36	130	70	2
Route 20	5.86	130	72	2
Route 21	6.48	130	72	2
	120.88	2821	1642	50

Table 3.6: 21 routes with a total cost of 1276.51 km

As mentioned earlier, the company has no reliable information about the current solution they are using to collect the waste in Seville. The only information provided is the number of kilometres travelled by the 20 vehicles over several days. Hence, in order to test the validity of the solutions obtained, we have estimated the total number of kilometres travelled. To this end, the mean of the total number of kilometres travelled on different days is used as the most appropriate estimator. Table 3.7

shows, for each number of vehicles, the best solution that the GRASP algorithm has obtained, the estimation, and the gap (i.e., the percentage deviation of the estimation value and the best solution that the GRASP algorithm has obtained).

	Proposed solution	Estimation	Gap
19 routes	1313.34 km	1450.29 km	10.39%
20 routes	1311.22 km	1438.38 km	7.41%
21 routes	1310.06 km	1426.46 km	8.89%

Table 3.7: Estimation of real routes

To conclude, if we take a look at Table 3.7, a mean reduction of more than 7% in the total distance is obtained independently of the number of routes. By using the proposed GRASP, not only are large savings achieved when considering the number of vehicles, but also for the total distance.

3.5.2 Multi-Objective Waste Collection Problem

In this section, the multi-objective GRASP used to solve the real-world WCP in Seville is presented. We have only adapted one out of the four multi-objective algorithms presented in Chapter 2 since, according to the performance in the literature instances, one algorithm attained notably better results; see the computational results from Section 2.4.2. According to these results, the best algorithm was called the **Ccomb LScomb GRASP**, which involves an ordered-sequential combined construction

and involves an ordered-sequential local search. The modifications needed to adapt the Ccomb LScomb GRASP in order to solve our real-world WCP in Seville are then laid out.

Again, one of the main difficulties involved in the WCP in Seville is the huge instance that we have to deal with, and hence only one out of the four algorithms has been adapted to solve the multi-objective WCP in Seville. To this end, and as happened in the single-objective case, only the construction phase needs to be adapted since the local search phase remains unchanged.

We have maintained the two algorithms in the construction phase (the extra-cost algorithm and the regret algorithm) of the GRASP for each objective function. Algorithm 9 shows the pseudo-code of the constructive procedure. As input parameters, the complete algorithm receives the transformed graph, $\hat{G}$, the number of vehicles, K, and the parameters for the RCLs, $\alpha^{(0)}$, $\alpha^{(1)}$, and $\alpha^{(2)}$ (line 1), and returns a set of non-dominated solutions (line 2). Both phases start by initializing the set of routes with the set of seeds (lines 4 to 9), by using the same greedy function, $g^{(0)}$, as in the single-objective GRASP.

Both construction algorithms are also based on the GRASP methodology (lines 10 to 34). In this case, four greedy functions are considered: $g_1^{(1)}$ and $g_1^{(2)}$, to optimize f_1; and $g_2^{(1)}$ and $g_2^{(2)}$, to optimize f_2.

- $g_1^{(1)}$ is the *extra-cost function* measured in units of length. It computes the change in the objective function when inserting an unserved street, j, at the best position, i, in the best route, k. In mathematical

terms, $g_1^{(1)}(j,k,i) = \hat{c}_{r_k(i-1)j} + \hat{c}_{jr_k(i)} - \hat{c}_{r_k(i-1)r_k(i)}$, where $r_k(i)$ denotes the vertex in the i-th position in route r_k (line 14).

- $g_2^{(1)}$ is the *extra-cost function for the longest route* measured in units of time. It computes the change in the objective function when inserting an unserved street, j, at the best position, i, in the longest route, k^*[1]. Formally, $g_2^{(1)}(j,k^*,i) = \hat{t}_{r_k^*(i-1)j} + \hat{t}_{jr_k^*(i)} - \hat{t}_{r_k^*(i-1)r_k^*(i)}$ where $r_k^*(i)$ denotes the i-th position of the longest route r_k^* (line 14).

- $g_1^{(2)}$ is the *regret function*. It computes the difference in distance when inserting each street with unserved containers, j, in the best route and in its second-best route (obviously in the best position). In mathematical terms, $g_1^{(2)}(j) = g_1^{(1)}(j,k^2,i_{k^2}) - g_1^{(1)}(j,k^1,i_{k^1})$, where k^1 denotes the route with the lowest extra mileage and k^2 denotes the route with the second-lowest extra mileage, i.e., we compute $k^1 = \arg\min_{k=1,\ldots,K} g_1^{(1)}(j,k,i)$ and $k^2 = \arg\min_{k=1,\ldots,K\setminus\{k^1\}} g_1^{(1)}(j,k,i)$ (line 27).

- $g_2^{(2)}$ is the *regret function for the longest route*. It computes the difference of time when inserting each street with unserved containers, j, in the best route (fixing this one as the longest route) and in its second best route (obviously in the best position). More specifically, $g_2^{(2)}(j) = g_2^{(1)}(j,k^{**},i_{k^{**}}) - g_2^{(1)}(j,k^*,i_{k^*})$, where k^* denotes the longest route and k^{**} denotes the route with the second-lowest extra cost,

[1] Again, to avoid cumbersome notation, we just consider k^* even whether the longest route is not the same for all the intermediate steps of the algorithm.

i.e., we compute $k^{**} = \arg \min_{k=1,\ldots,K\setminus\{k^*\}} g_2^{(1)}(j,k,i)$ (line 27).

As is customary in a GRASP design, the algorithm constructs the RCL, selects an element at random, and updates the affected variables. This strategy adds elements to routes, on the condition that the capacity of a vehicle or the working-day constraint is not exceeded.

Again, due to the proposed GRASP constructive does not guarantee the feasibility of the obtained solution, when the solution is unfeasible, the solution is discarded and the procedure starts to construct new solutions. Nevertheless, these details have not been included in the pseudocode of Algorithm 9.

Finally, as always in multi-objective optimization, a set, R, of non-dominated solutions must be considered. Thus, every feasible solution entering the construction phase has to be checked for its possible inclusion in R. When a new solution is admitted into R, it then has to be checked whether this new solution dominates other solutions in R. This process is represented with the function $Insert\&Update$ (line 22 and line 35).

Algorithm 9 Multi-objective Construction Phase (ordered-sequential combined)

1: **Input:** $\hat{G} = (\hat{V}, \hat{A})$, K, $\alpha^{(0)}$, $\alpha^{(1)}$, $\alpha^{(2)}$;
2: **Output:** $R = \{(r = (r_k)_{k=1,\dots,K})\}$ set of non-dominated solutions;
3: **Initialize:** $S = \{0, l\}$, $J = \hat{V} \setminus \{0, l\}$, $k \leftarrow 1$, $R \leftarrow \emptyset$;
4: **repeat**
5: For all $j \in J$ compute $g^{(0)}(j)$;
6: Define $RCL^{(0)} \leftarrow \{j \in J | g^{(0)}(j) \geq g^{(0)}_{max} - \alpha^{(0)}(g^{(0)}_{max} - g^{(0)}_{min})\}$;
7: Select j^* at random from $RCL^{(0)}(J)$;
8: $r_k \leftarrow r_k \cup \{j^*\}$, $S \leftarrow S \cup \{j^*\}$, $J \leftarrow J \setminus \{j^*\}$, $k \leftarrow k + 1$;
9: **until** $k = K + 1$;
10: $r^{(1)} \leftarrow r$, $J^{(1)} \leftarrow J$;
11: **while** $J^{(1)} \neq \emptyset$ **do**
12: **while** the capacity or the working day of the K vehicles is not exceeded **do**
13: **for all** Objective $m = 1, 2$ **do**
14: For all $j \in J^{(1)}$ compute $g^{(1)}_m(j, k, i)$;
15: Define $RCL^{(1)}_m \leftarrow \{j \in J^{(1)} | g^{(1)}_m(j, k, i) \leq g^{(1)}_{m,min} + \alpha^{(1)}(g^{(1)}_{m,max} - g^{(1)}_{m,min})\}$;
16: Select j^* at random from $RCL^{(1)}_m(J^{(1)})$;
17: $r^{(1)} \leftarrow r^{(1)} \cup \{j^*\}$, $J^{(1)} \leftarrow J^{(1)} \setminus \{j^*\}$;
18: **end for**
19: **end while**
20: Go to line 4, $S = \{l\}$, $J \leftarrow J^{(1)}$, $k \leftarrow 1$;
21: **end while**
22: $R \leftarrow Insert\&Update(r^{(1)})$;
23: $r^{(2)} \leftarrow r$, $J^{(2)} \leftarrow J$;
24: **while** $J^{(2)} \neq \emptyset$ **do**
25: **while** the capacity or the working day of the K vehicles is not exceeded **do**
26: **for all** Objective $m = 1, 2$ **do**
27: For all $j \in J^{(2)}$ compute $g^{(2)}_m(j)$;
28: Define $RCL^{(2)}_m \leftarrow \{j \in J^{(2)} | g^{(2)}_m(j) \geq g^{(2)}_{m,max} - \alpha^{(2)}(g^{(2)}_{m,max} - g^{(2)}_{m,min})\}$;
29: Select j^* at random from $RCL^{(2)}_m(J^{(2)})$;
30: $r^{(2)} \leftarrow r^{(2)} \cup \{j^*\}$, $J^{(2)} \leftarrow J^{(2)} \setminus \{j^*\}$;
31: **end for**
32: **end while**
33: Go to line 4, $S = \{l\}$, $J \leftarrow J^{(2)}$, $k \leftarrow 1$;
34: **end while**
35: $R \leftarrow Insert\&Update(r^{(2)})$;
36: **return** R

Computational Results

The results obtained when applying the methodology described below to the WCP in Seville are now described.

As we did in the single-objective WCP in Seville, the number of vehicle routes is considered as an input of the problem. We solve the real-world problem by taking three values of the number of vehicles in order to provide the company with a variety of solutions.

The proposed GRASP limits the CPU time to $90,000$ seconds for each number of vehicles. Furthermore, the seeds are chosen in the GRASP by randomly taking $\alpha^{(0)}$, $\alpha^{(1)} = 0.01$, and $\alpha^{(2)} = 0.01$.

Table 3.8 presents the set of non-dominated solutions obtained by applying the multi-objective GRASP to solve the WCP in Seville.

Figures 3.6, 3.7, and 3.8 present the set of non-dominated solutions obtained solving the WCP in Seville for 19, 20, and 21 vehicles, respectively. Specifically, the x-axis represents the total number of kilometres travelled by all vehicles and the y-axis represents the duration of the longest route, that is, the trade-off between the two conflicting objectives.

In view of the results of Tables 3.8 and of Figures 3.6, 3.7, and 3.8, it can be concluded that the greater the total distance travelled by all vehicles, the more balanced the duration of the routes.

Finally, Figure 3.9 shows a comparison between the three Pareto fronts. Note that those Pareto fronts are not strictly compared since each front refers to a different number of vehicles.

Routes	Total distance (km)	Longest time of a route (hours)
19	1480.83	6.96
19	1483.20	6.93
20	1489.81	6.84
20	1489.12	6.88
20	1467.38	6.91
20	1466.36	6.97
20	1466.23	6.98
20	1390.11	6.99
20	1387.37	7.04
21	1475.71	6.59
21	1475.06	6.59
21	1470.03	6.61
21	1460.90	6.62
21	1442.92	6.73
21	1433.31	6.76
21	1428.29	6.94
21	1427.49	6.96

Table 3.8: Set of non-dominated solutions for the WCP in Seville.

All solutions obtained solving the multi-objective WCP in Seville will be provided to the company so that they can choose the Pareto optimal solution in accordance with their preferences.

For the multi-objective WCP in Seville, no visual impression of the obtained solutions is given (as we did in Appendix A) since the quantity of obtained routes is very high ($19 \times 2 + 20 \times 7 + 21 \times 8 = 346$).

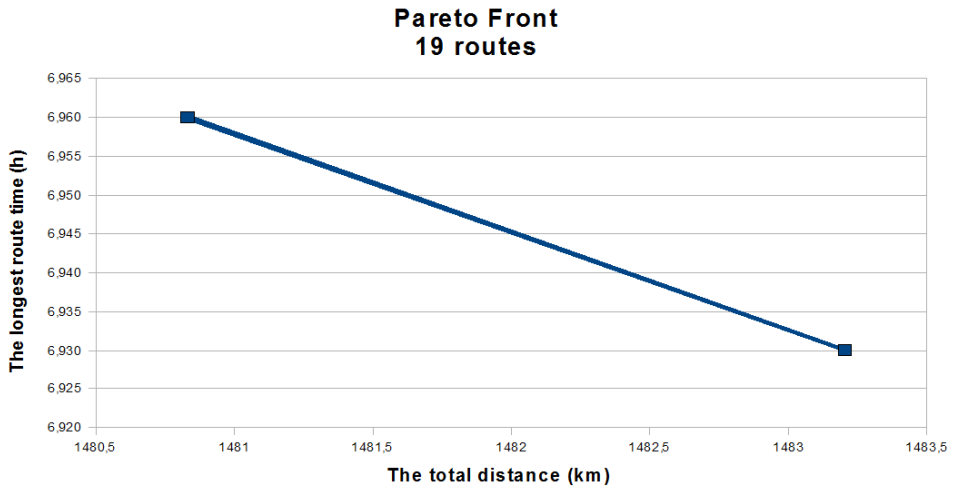


Figure 3.6: Pareto front obtained solving the WCP in Seville for 19 vehicles

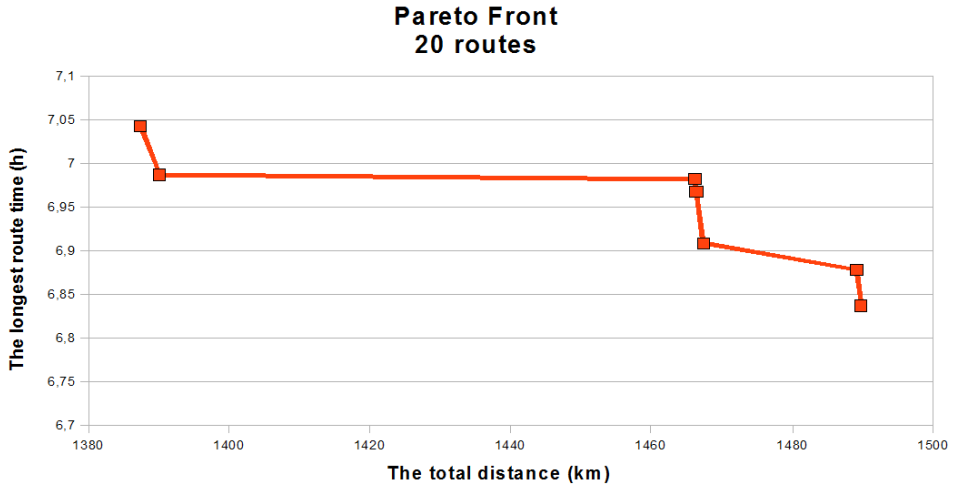


Figure 3.7: Pareto front obtained solving the WCP in Seville for 20 vehicles

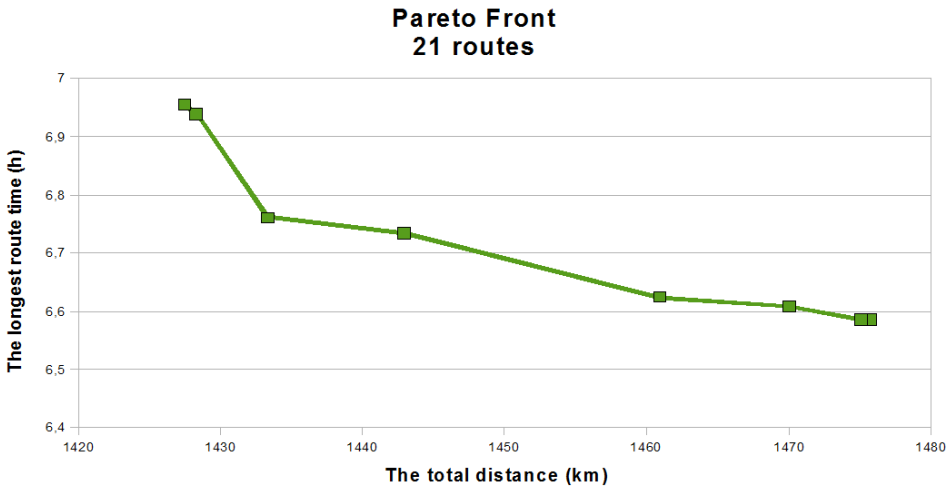


Figure 3.8: Pareto front obtained solving the WCP in Seville for 21 vehicles

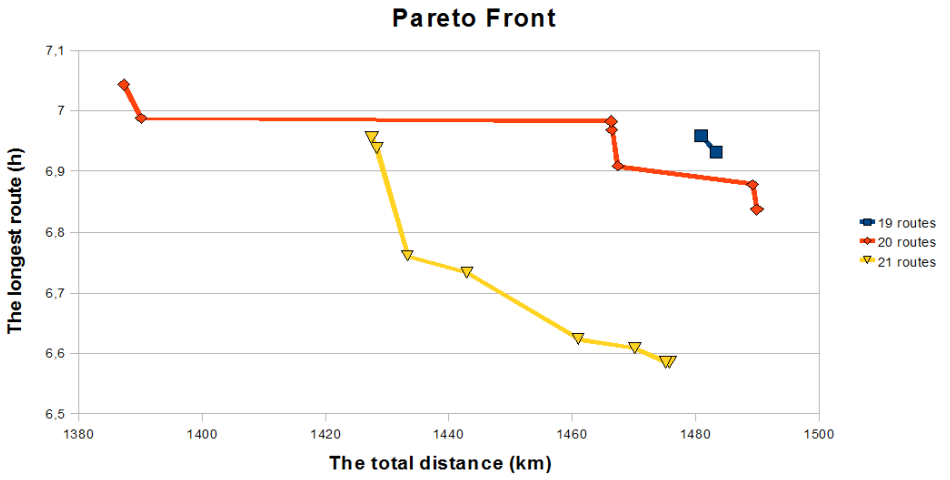


Figure 3.9: Comparison of the three Pareto fronts obtained solving the WCP in Seville

Conclusions and Future Research

This research was originally motivated by a real-world problem, as proposed by the local authority responsible for the collection of urban waste in Seville, Spain. As a consequence, in this Ph.D. thesis, we have devised various approaches to solve the waste collection problem in Seville in order to attain an improvement of the planning of this service in the city.

The main contributions of this dissertation include: the proposed algorithm for the solution of single-objective routing problems, whose objective is to minimize the total routing cost; and the four proposed algorithms for the solution of multi-objective routing problems, in which a second objective function is introduced in the form of, balancing the routes. All algorithms presented use a hybrid method that combines GRASP constructions with VND local searches. It is worth mentioning that the construction phase and the local search phase differ in accordance with which objective function is being evaluated at the time. Furthermore and to the best of our knowledge, the local search results in a new set of neighbourhood structures specifically designed to optimize the second

objective; hitherto unknown in the literature. These algorithms are used since they are easy to understand, implement, and are also flexible to adaptation to more complex situations, or even the inclusion of new side constraints. The proposed algorithms are valid for the solution of both DCARPs and ACVRPs. Nevertheless, computational results are tested only for DCARP instances, since the real-world problem of interest in this Ph.D. thesis shares more similarities with the DCARP than with the ACVRP.

The algorithms are evaluated in 56 instances of up to 50 nodes, up to 194 arcs and up to 97 required arcs. For the single-objective version of the considered problem, we point out that in 3 out of the 56 problems, the proposed algorithm improves the best solutions found to date, in 31 out of the 56 problems, the algorithm obtains the same solutions, and finally, in 22 out of the 56 problems the algorithm remains unable to find the best solutions, although in most cases this gap is small. Furthermore, in order to demonstrate the validity of our algorithm against similar algorithms, a statistical test is performed. The results of the analysis indicate that, on average, all algorithms obtain similar results. For the multi-objective version of the considered problem, no algorithms are found that solve DCARPs or ACVRPs: only undirected versions of the problem. Hence, in order to illustrate the performance of the algorithms for routing problems on directed graphs, our four algorithms are compared against each other.

Finally, the proposed algorithms for the solution of routing problems in general, are adapted to solve the waste collection in Seville by introducing

several slight variations. Once again, both versions of the problem are considered: the waste collection, where only the minimization of the total distance is considered; and the waste collection, where the total distance is optimized, and the routes are balanced in order to equalize their working times. By solving the waste collection in Seville, significant savings are achieved on both number of vehicles and total distance travelled, which implies large savings in the annual budget for solid waste collection per year in this city. Specifically, by using 19 vehicles (one vehicle fewer), the mean reduction was 10.39% in the total distance travelled by all vehicles; by using 20 vehicles, the mean reduction is 7.41%; and by using 21 vehicles (one vehicle more), the mean reduction is 8.89%. By including the balance of the routes, the savings cannot be quantified, although there must be greater satisfaction on the part of the workers since now the duration of the working day is much more similar.

Before concluding, it is worth emphasize that the work performed in this Ph.D. thesis has opened other new lines of research related with the waste collection problem.

An interesting problem related to this work is the waste collection for each kind of disposal (green containers, blue containers, and yellow containers) that must be collected separately by each vehicle, and whose frequency of collection is not necessarily daily. This problem can be modelled as a Period Vehicle Routing Problem with Service Choice, which is a variation of the Period Vehicle Routing Problem in which service frequency and day combinations are decisions of the model. In particular,

our Period Vehicle Routing Problem with Service Choice requires frequencies and day combinations to be assigned to containers, and also vehicles per day and type of waste to be assigned in order to minimize the number of vehicles combined with routing which, in turn, minimizes the total distance. Note that, if a container is associated with a higher service frequency, the routing cost increases, since this site has to be visited more often. It is therefore assumed that each container requires a minimum number of visits per period in accordance with demand, i.e., a preset frequency. These concepts are further developed by Francis and Smilowitz, see [53], who study an application arising in interlibrary loan delivery services.

In the future, a very interesting and ambitious project, would be to build a software tool, that enables interaction between the GIS and the algorithm. In this way, we would have the advantage of working with the most distinguished characteristics of both: on the one hand, the GIS, which allows us to store, modify, retrieve, analyze and visualize the data information; and on the other hand, the algorithm that solves the problem in a very short time, and include all the necessary features. Furthermore, we could strive towards the inclusion of certain improvements in the algorithms and, in turn, towards their inclusion into the software. For instance, the local search of the proposed algorithms could be optimized. Specifically, we should present a strategy for the reduction of the computational complexity required for the application of our local search. Hence, implementations of a more efficient nature could be designed, and the

local search could be facilitated in practical instances of a very large size, such as in our real-world instance. The main concept for the reduction in complexity involves limiting the number of movements to evaluate, i.e., it reduces the size of the neighbourhoods that are explored. Further to this, other objectives could be included, and finally, comparisons between popular multi-objective algorithms, such as the NSGA-II, could also be included.

Another line of future research could involve cost allocation. In Seville, citizens pay taxes for the waste collection in terms of the value of their house. Nevertheless, the theory of game routing problems (see Potters *et al.* (1992) as a seminar paper on this theory) could be applied to find other reasonable and fair cost allocations of these taxes.

Appendix A

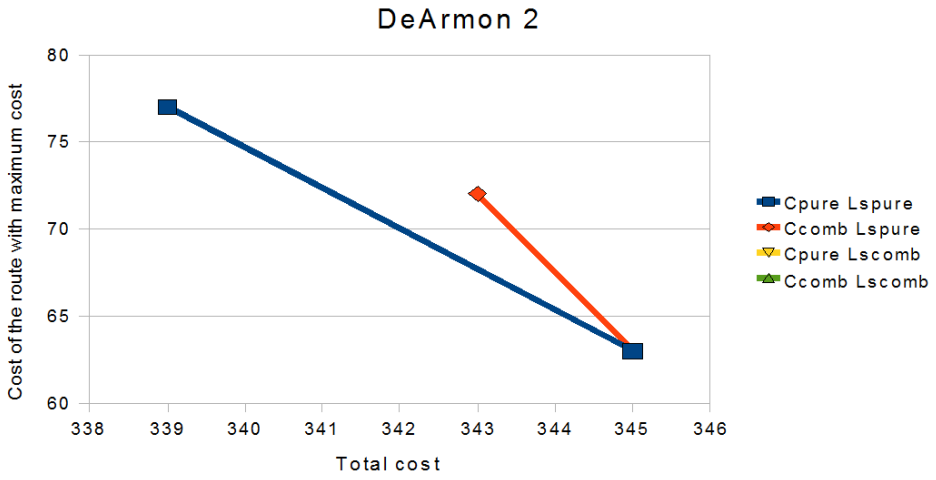
Computational Results for the Multi-Objective Problem

Pareto-optimal solutions on the DeArmon 1

	Cpure LSpure		Ccomb LSpure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA1	323	66	323	66	323	66	323	66
	319	74	319	74	319	74	319	74
	316	77	316	77	316	77	316	77

Pareto-optimal solutions on the DeArmon 2

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA2	345	63	345	63	345	63	345	63
	339	77	343	72	339	77		



Pareto-optimal solutions on the DeArmon 3

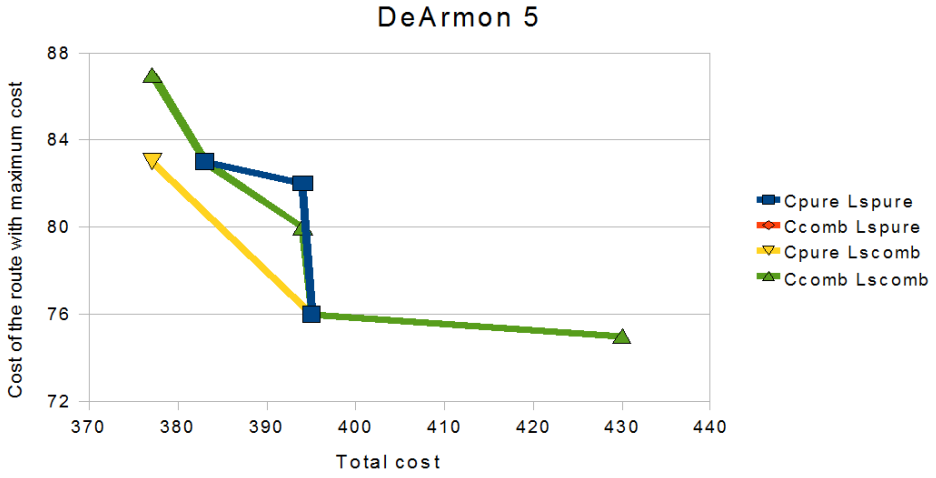
	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA3	289	61	289	61	289	61	289	61
	275	66	275	66	275	66	275	66

Pareto-optimal solutions on the DeArmon 4

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA4	292	74	292	74	292	74	292	74
	287	81	287	81	287	81	287	81

Pareto-optimal solutions on the DeArmon 5

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA5	395	76	395	76	395	76	430	75
	394	82	394	82	377	83	395	76
	383	83	383	83			394	80
							383	83
							377	87



DA5	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.250	0.063	0.188	0.063
Hypervolume	0.234	0.033	0.152	0.033
R_2 Indicator	0.075	0.002	0.056	0.002

DA5 Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.000	0.000	0.000
Ccomb LSpure	0.000	-	0.000	0.000
Cpure LScomb	0.333	0.333	-	0.400
Ccomb LScomb	0.333	0.333	0.000	-

Pareto-optimal solutions on the DeArmon 6

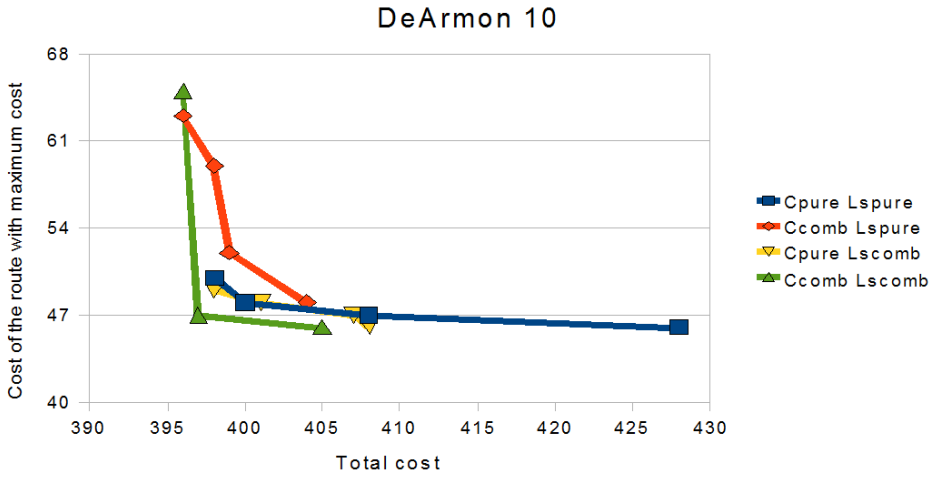
	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA6	319	68	319	68	319	68	319	68
	313	73	313	73	313	73	313	73
	298	75	298	75	298	75	298	75

Pareto-optimal solutions on the DeArmon 7

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA7	325	68	325	68	325	68	325	68

Pareto-optimal solutions on the DeArmon 10

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA10	428	46	404	48	408	46	405	46
	408	47	399	52	407	47	397	47
	400	48	398	59	401	48	396	65
	398	50	396	63	398	49		

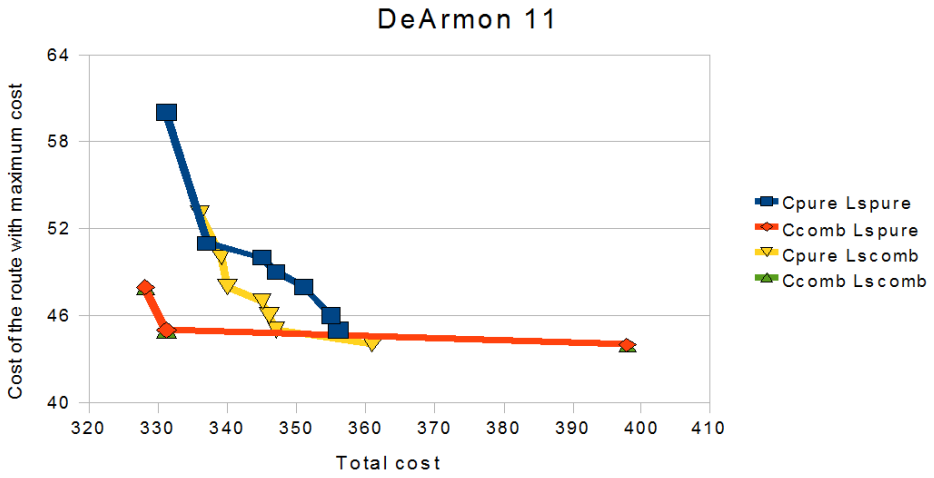


DA10	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.000	0.033	0.033	0.033
Hypervolume	0.000	0.005	0.008	0.008
R_2 Indicator	0.000	0.000	0.001	0.001

DA10 Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.750	0.250	0.000
Ccomb LSpure	0.000	-	0.000	0.333
Cpure LScomb	0.750	0.750	-	0.000
Ccomb LScomb	1.000	0.750	1.000	-

Pareto-optimal solutions on the DeArmon 11

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA11	356	45	398	44	361	44	398	44
	355	46	331	45	347	45	331	45
	351	48	328	48	346	46	328	48
	347	49			345	47		
	345	50			340	48		
	337	51			339	50		
	331	60			336	53		

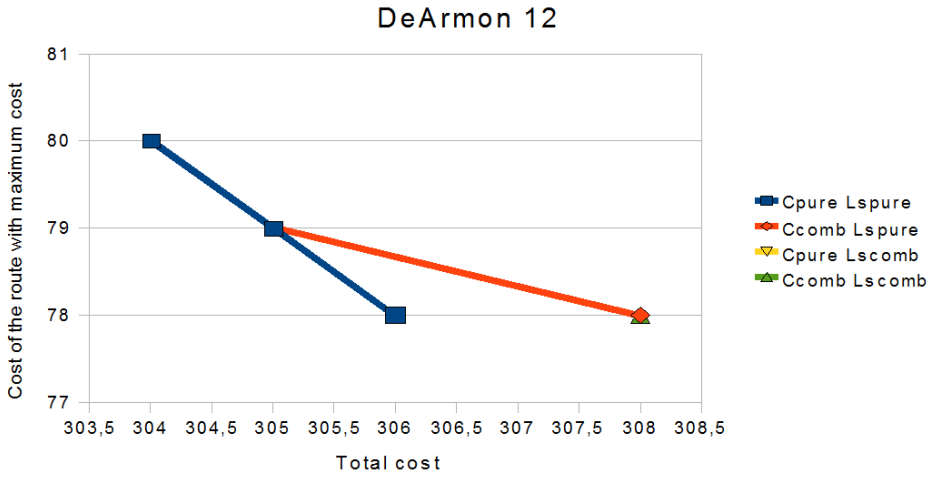


DA11	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.417	0.333	0.333	0.100
Hypervolume	0.183	0.168	0.033	0.033
R_2 Indicator	0.091	0.076	0.043	0.005

DA11 Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.000	0.000	0.000
Ccomb LSpure	1.000	-	0.857	0.000
Cpure LScomb	0.714	0.333	-	0.333
Ccomb LScomb	1.000	0.000	0.857	-

Pareto-optimal solutions on the DeArmon 12

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA12	306	78	308	78	305	79	308	78
	305	79	305	79	304	80	305	79
	304	80	304	80			304	80

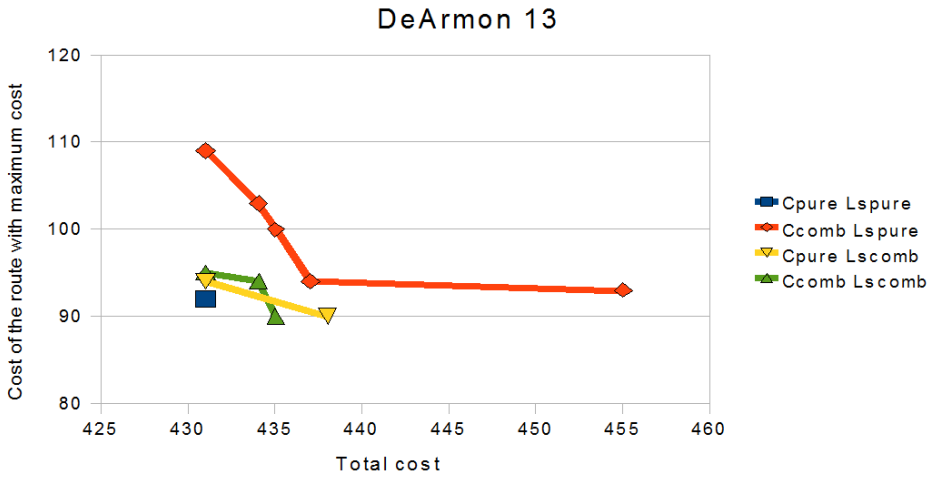


DA12	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.643	0.172	0.643	0.000
Hypervolume	0.766	0.253	0.776	0.000
R_2 Indicator	0.278	0.065	0.271	0.000

DA12 Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.333	0.000	0.333
Ccomb LSpure	0.000	-	0.000	0.000
Cpure LScomb	0.000	0.000	-	0.000
Ccomb LScomb	0.000	0.000	0.000	-

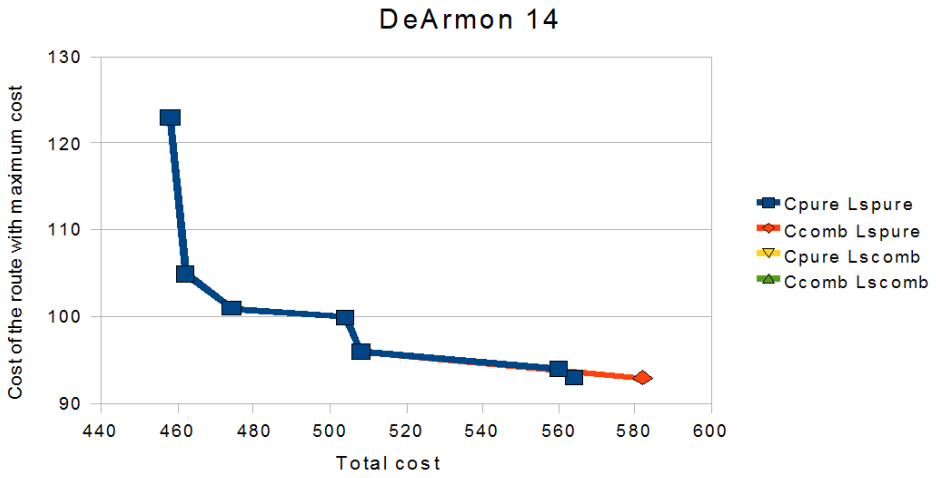
Pareto-optimal solutions on the DeArmon 13

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA13	431	92	455	93	438	90	435	90
			437	94	431	94	434	94
			435	100			431	95
			434	103				
			431	109				



Pareto-optimal solutions on the DeArmon 14

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA14	564	93	582	93	560	94	560	94
	560	94	508	96	508	96	508	96
	508	96	504	100	504	100	504	100
	504	100	474	101	474	101	474	101
	474	101	462	105	462	105	462	105
	462	105	458	123	458	123	458	123
	458	123						

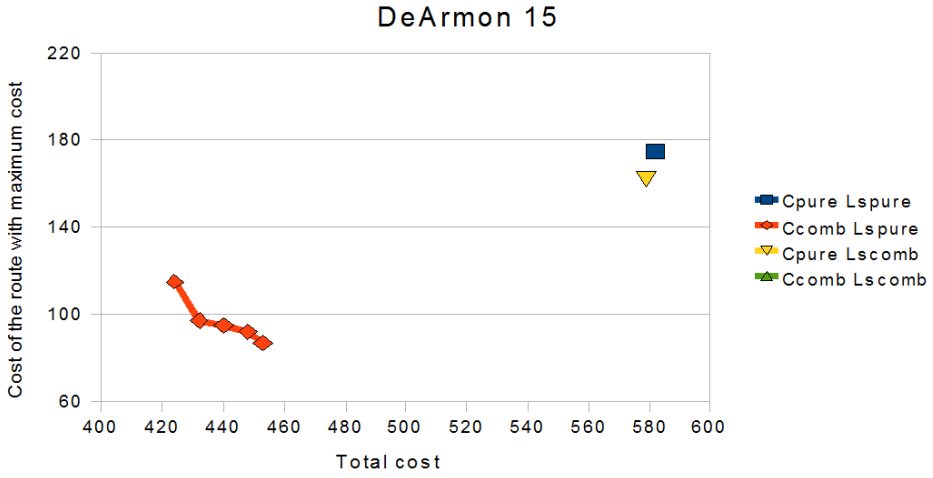


DA14	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.000	0.000	0.000	0.000
Hypervolume	0.000	0.000	0.000	0.000
R_2 Indicator	0.000	0.000	0.000	0.000

DA14 Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.143	0.000	0.000
Ccomb LSpure	0.000	-	0.000	0.000
Cpure LScomb	0.000	0.000	-	0.000
Ccomb LScomb	0.000	0.000	0.000	-

Pareto-optimal solutions on the DeArmon 15

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA15	582	175	453	87	579	162		
			448	92				
			440	95				
			432	97				
			424	115				



Pareto-optimal solutions on the DeArmon 16

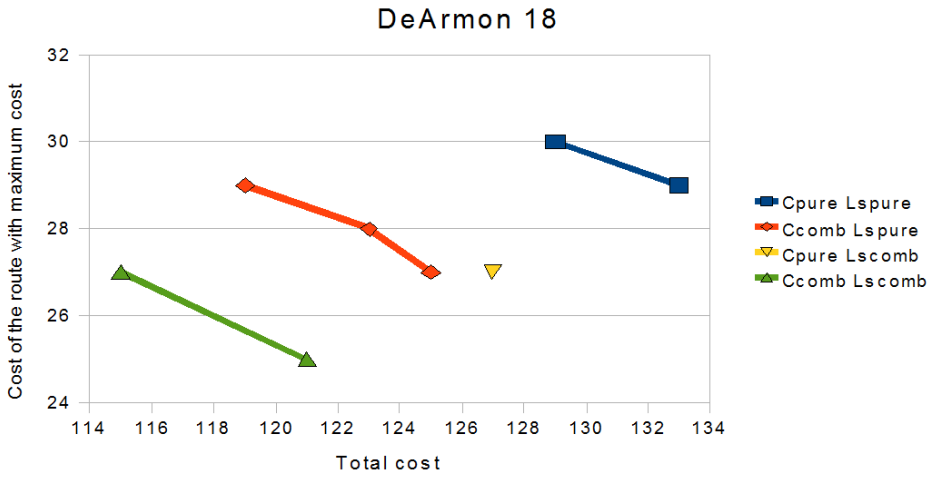
	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA16	100	23	100	23	100	23	100	23

Pareto-optimal solutions on the DeArmon 17

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA17	58	15	58	15	58	15	58	15

Pareto-optimal solutions on the DeArmon 18

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA18	133	29	125	27	127	27	121	25
	129	30	123	28			115	27
			119	29				

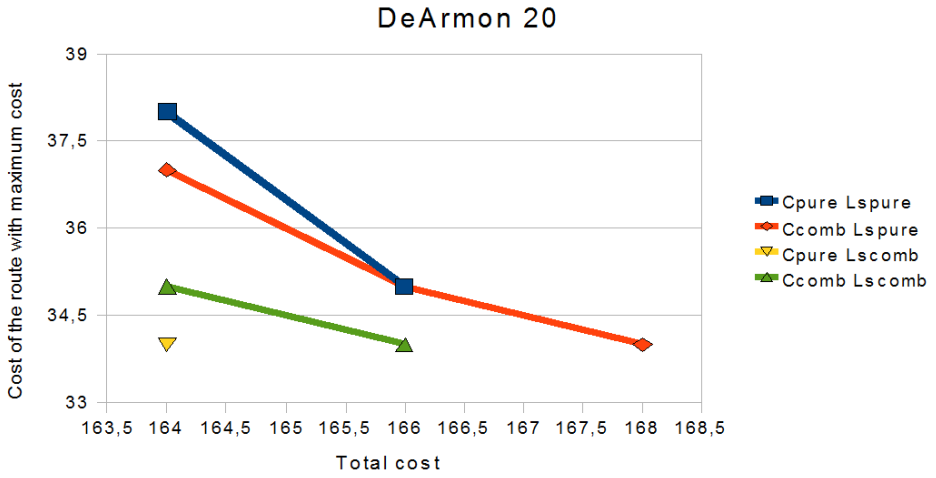


Pareto-optimal solutions on the DeArmon 19

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA19	91	19	91	19	91	19	91	19

Pareto-optimal solutions on the DeArmon 20

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA20	166	35	168	34	164	34	166	34
	164	38	166	35			164	35
			164	37				

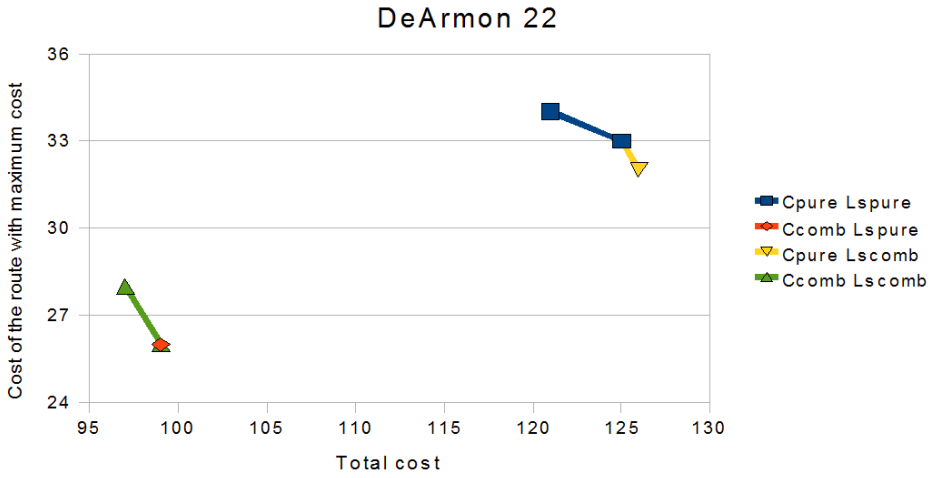


Pareto-optimal solutions on the DeArmon 21

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA21	57	22	57	22	57	22	57	22

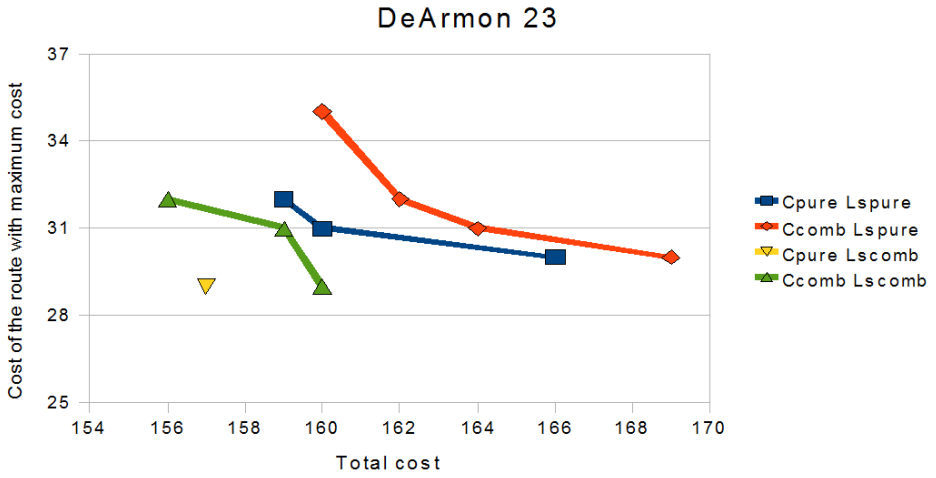
Pareto-optimal solutions on the DeArmon 22

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA22	125	33	99	26	126	32	99	26
	121	34			125	33	97	28
					121	34		



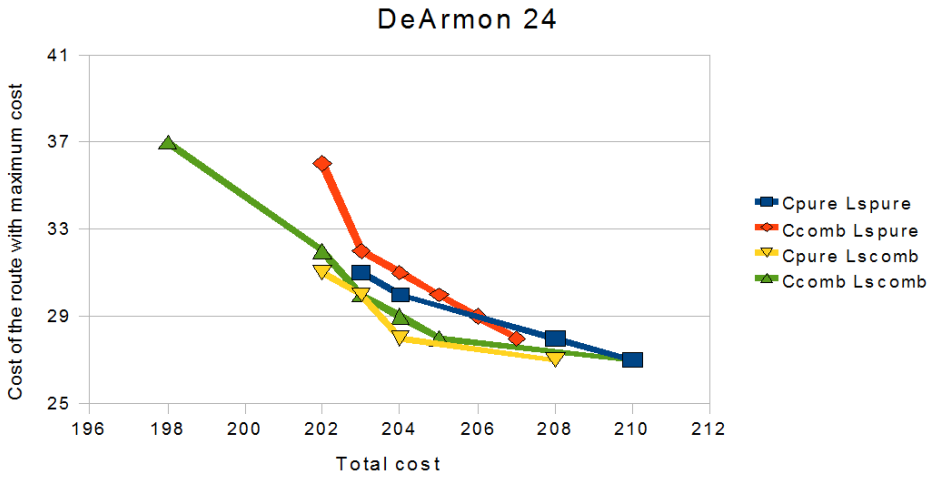
Pareto-optimal solutions on the DeArmon 23

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA23	166	30	160	35	157	29	160	29
	160	31	162	32			159	31
	159	32	164	31			156	32
			169	30				



Pareto-optimal solutions on the DeArmon 24

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA24	210	27	207	28	208	27	210	27
	208	28	206	29	204	28	205	28
	204	30	205	30	203	30	204	29
	203	31	204	31	202	31	203	30
			203	32			202	32
			202	36			198	37

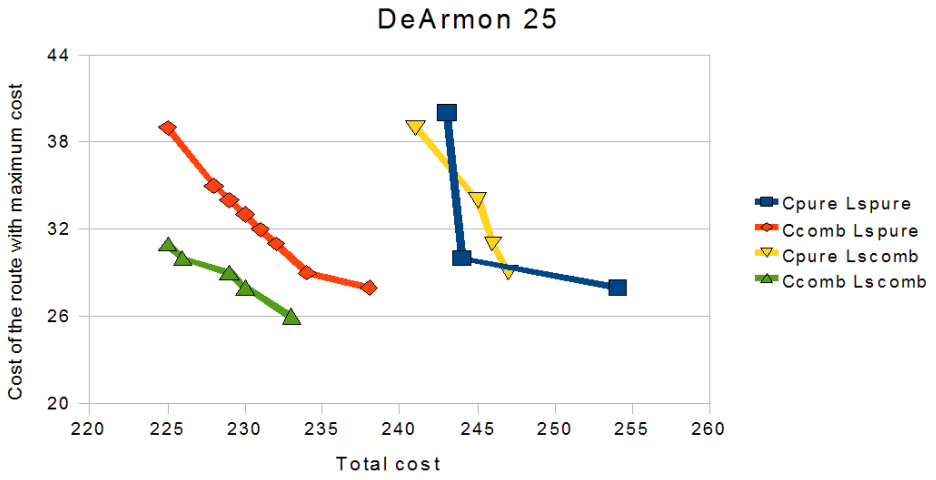


DA24	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.113	0.113	0.083	0.113
Hypervolume	0.061	0.061	0.013	0.038
R_2 Indicator	0.014	0.014	0.005	0.003

DA24 Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.500	0.000	0.000
Ccomb LSpure	0.250	-	0.000	0.000
Cpure LScomb	1.000	1.000	-	0.000
Ccomb LScomb	0.750	1.000	0.000	-

Pareto-optimal solutions on the DeArmon 25

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
DA25	254	28	238	28	247	29	233	26
	244	30	234	29	246	31	230	28
	243	40	232	31	245	34	229	29
			231	32	241	39	226	30
			230	33			225	31
			229	34				
			228	35				
			225	39				



DA25	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.000	0.000	0.000	0.000
Hypervolume	0.000	0.000	0.000	0.000
R_2 Indicator	0.000	0.000	0.000	0.000

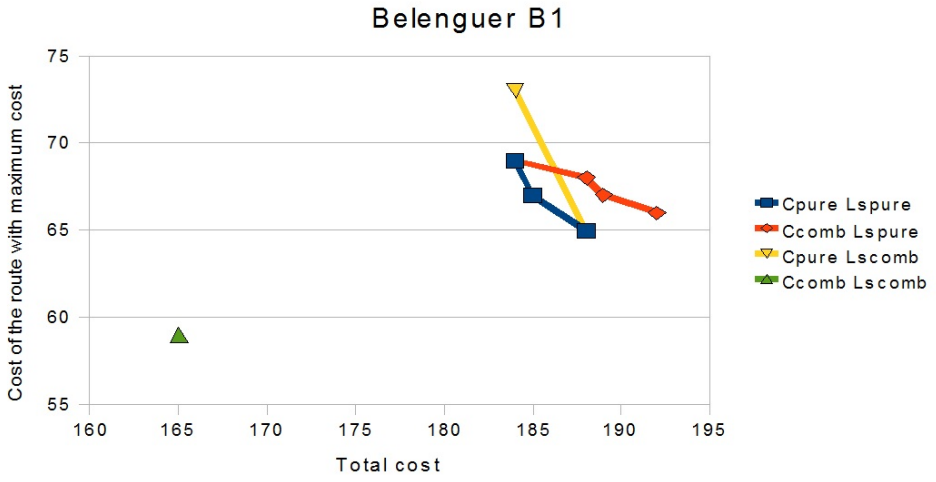
DA25 Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.500	0.000	0.000
Ccomb LSpure	1.000	-	1.000	0.000
Cpure LScomb	0.333	0.000	-	0.000
Ccomb LScomb	1.000	1.000	1.000	-

Pareto-optimal solutions on the Belenguer 1A

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
1A	173	87	173	87	173	87	173	87

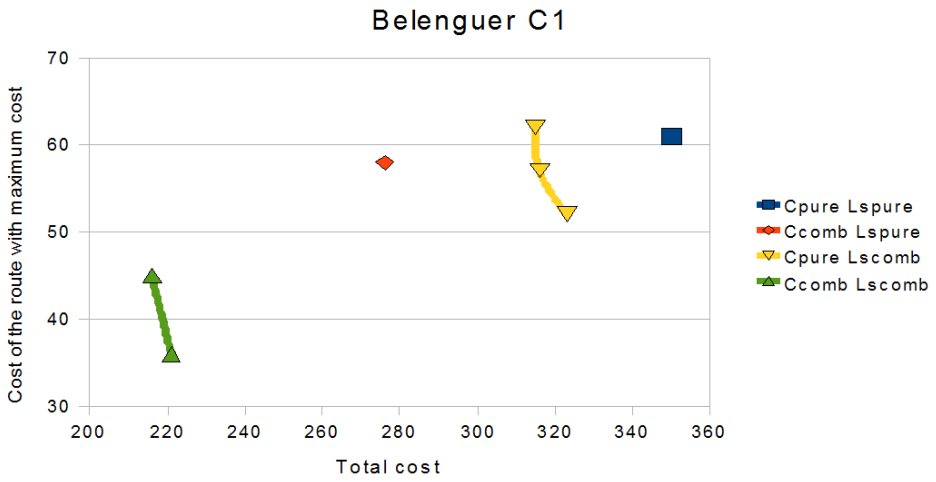
Pareto-optimal solutions on the Belenguer 1B

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
1B	188	65	192	66	188	65	165	59
	185	67	189	67	184	73		
	184	69	188	68				
			184	69				



Pareto-optimal solutions on the Belenguer 1C

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
1C	350	61	276	58	323	52	216	45
					316	57	221	36
					315	62		

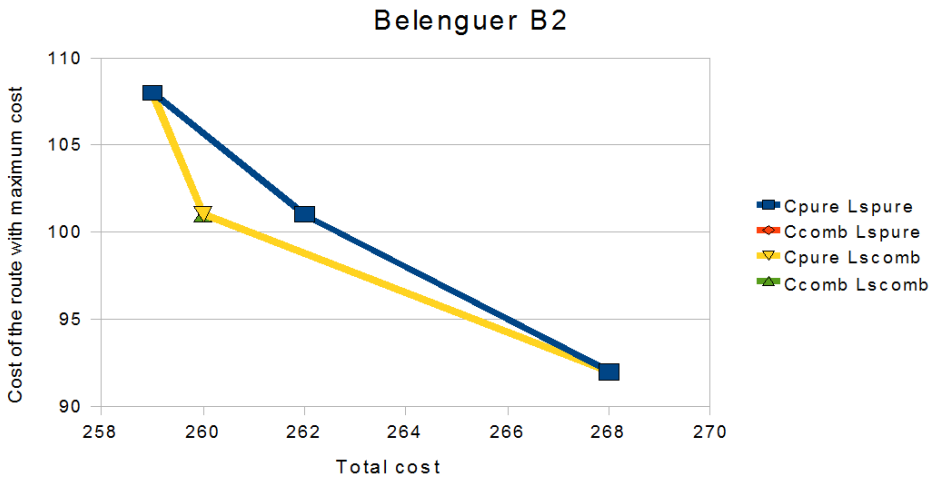


Pareto-optimal solutions on the Belenguer 2A

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
2A	229	115	229	115	229	115	229	115
	227	117	227	117	227	117	227	117

Pareto-optimal solutions on the Belenguer 2B

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
2B	268	92	268	92	268	92	268	92
	262	101	262	101	260	101	260	101
	259	108	259	108	259	108	259	108

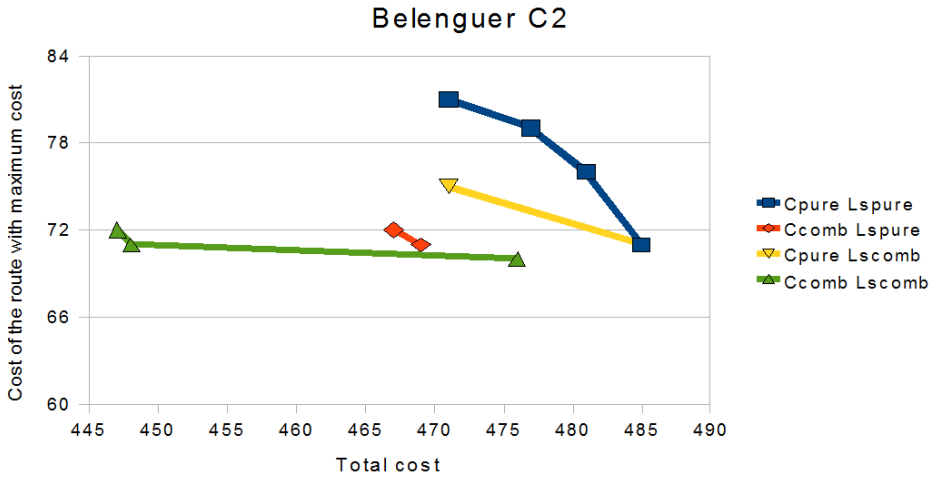


2B	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.000	0.000	0.000	0.000
Hypervolume	0.000	0.000	0.000	0.000
R_2 Indicator	0.000	0.000	0.000	0.000

2B Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.000	0.000	0.000
Ccomb LSpure	0.000	-	0.000	0.000
Cpure LScomb	0.333	0.333	-	0.000
Ccomb LScomb	0.333	0.333	0.000	-

Pareto-optimal solutions on the Belenguer 2C

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
2C	485	71	469	71	485	71	476	70
	481	76	467	72	471	75	448	71
	477	79					447	72
	471	81						



2C	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.222	0.222	0.000	0.000
Hypervolume	0.097	0.097	0.000	0.000
R_2 Indicator	0.017	0.017	0.000	0.000

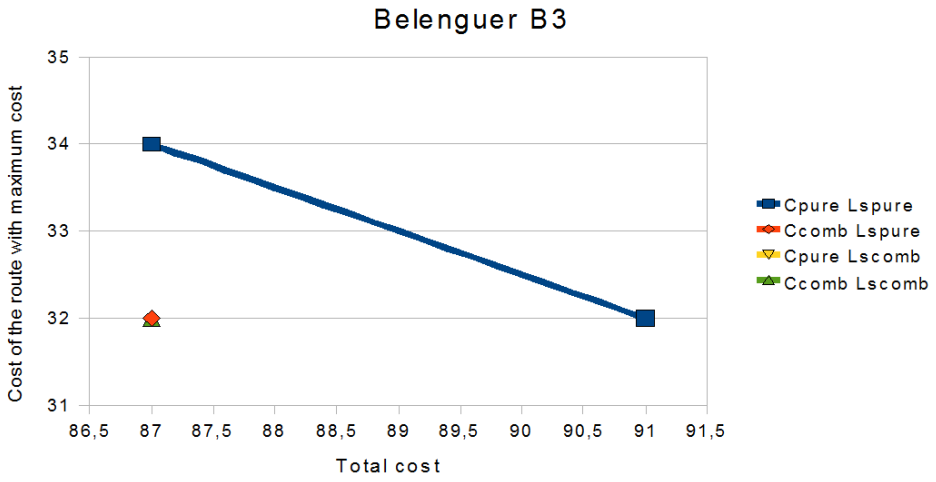
2C Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.000	0.000	0.000
Ccomb LSpure	1.000	-	1.000	0.000
Cpure LScomb	0.750	0.000	-	0.000
Ccomb LScomb	1.000	1.000	1.000	-

Pareto-optimal solutions on the Belenguer 3A

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
3A	81	41	81	41	81	41	81	41

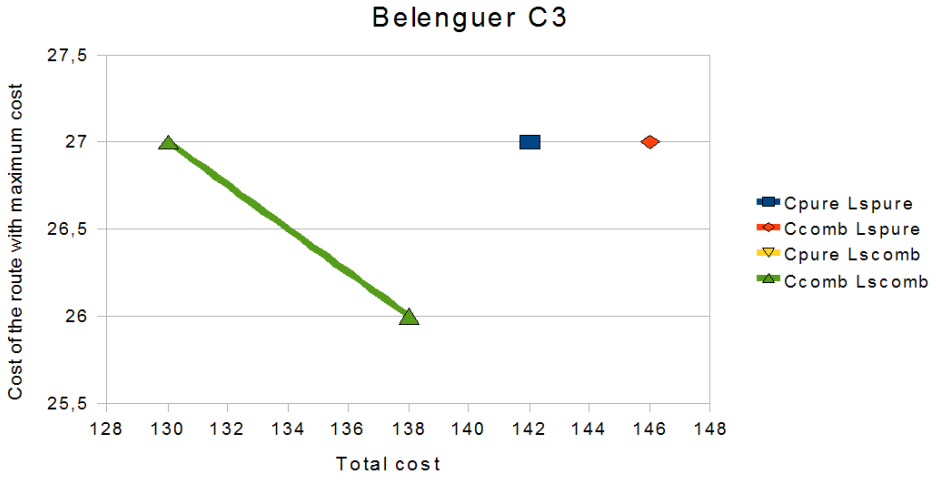
Pareto-optimal solutions on the Belenguer 3B

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
3B	91	32	87	32	91	32	87	32
	87	34			87	34		



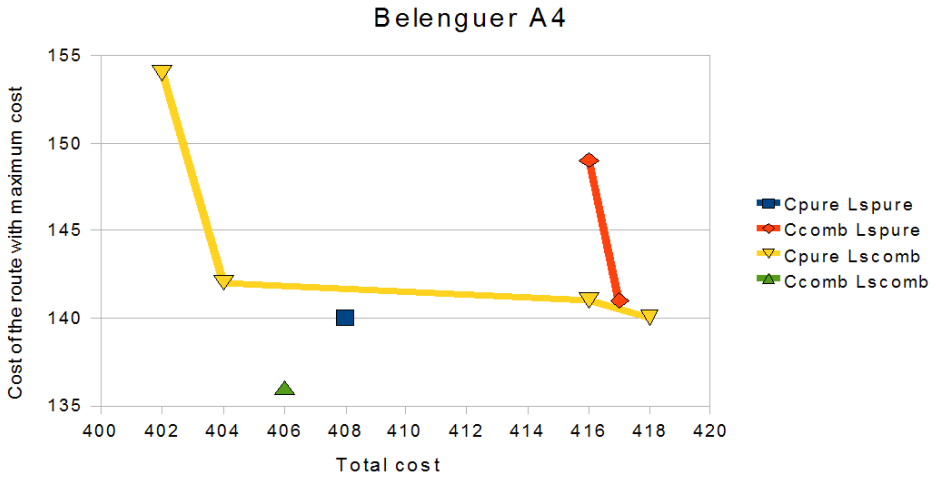
Pareto-optimal solutions on the Belenguer 3C

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
3C	142	27	146	27	142	27	138	26
							130	27



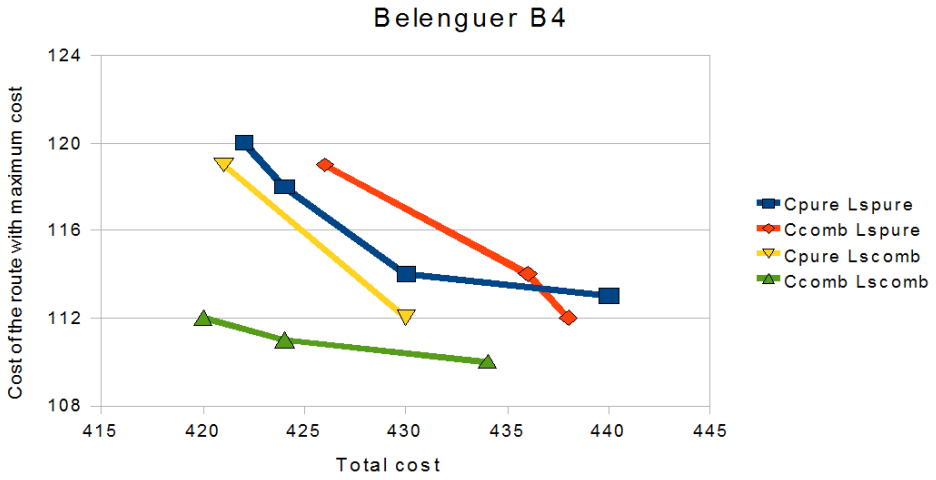
Pareto-optimal solutions on the Belenguer 4A

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
4A	408	140	417	141	418	140	406	136
			416	149	416	141		
					404	142		
					402	154		



Pareto-optimal solutions on the Belenguer 4B

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
4B	440	113	438	112	430	112	434	110
	430	114	436	114	421	119	424	111
	424	118	426	119			420	112
	422	120						

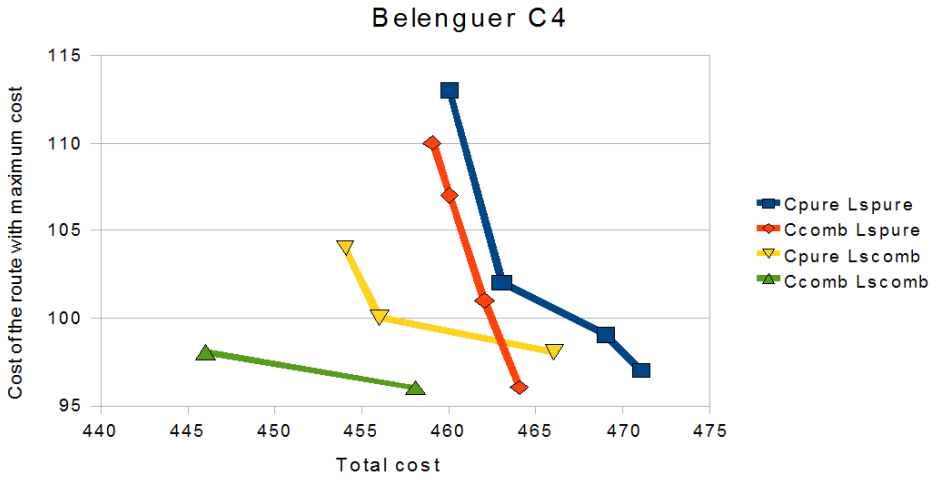


4B	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.789	0.526	0.632	0.000
Hypervolume	0.933	0.564	0.800	0.000
R_2 Indicator	0.324	0.229	0.301	0.000

4B Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.667	0.000	0.000
Ccomb LSpure	0.750	-	0.000	0.000
Cpure LScomb	0.250	1.000	-	0.000
Ccomb LScomb	1.000	1.000	1.000	-

Pareto-optimal solutions on the Belenguer 4C

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
4C	471	97	464	96	466	98	458	96
	469	99	462	101	456	100	446	98
	463	102	460	107	454	104		
	460	113	459	110				

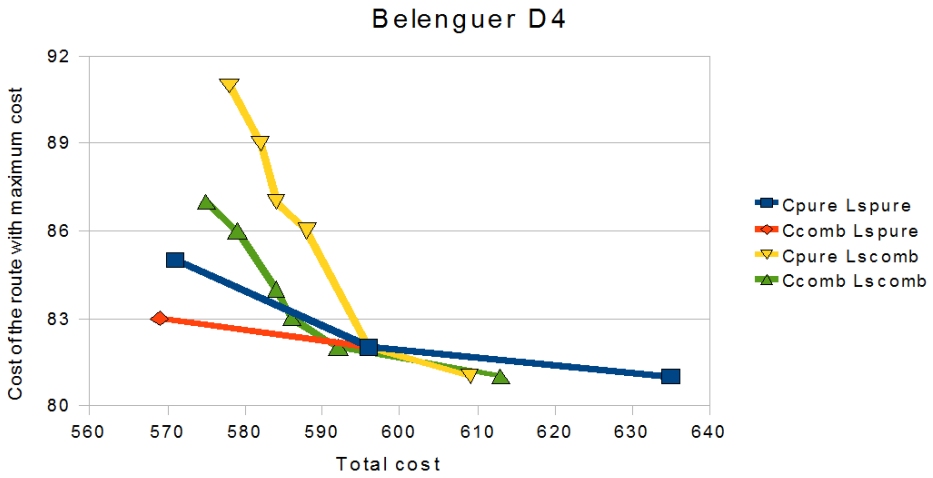


4C	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.500	0.700	0.500	0.000
Hypervolume	0.590	0.770	0.490	0.000
R_2 Indicator	0.169	0.214	0.131	0.000

4C Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.000	0.000	0.000
Ccomb LSpure	1.000	-	0.333	0.000
Cpure LScomb	0.750	0.750	-	0.000
Ccomb LScomb	1.000	1.000	1.000	-

Pareto-optimal solutions on the Belenguer 4D

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
4D	635	81	596	82	609	81	613	81
	596	82	569	83	596	82	592	82
	571	85			588	86	586	83
					584	87	584	84
					582	89	579	86
					578	91	575	87

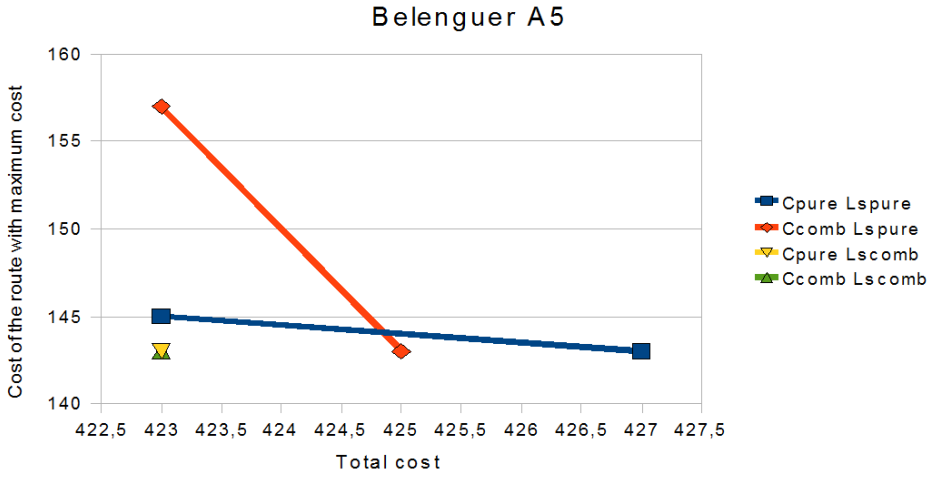


4D	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.680	0.560	0.353	0.000
Hypervolume	0.784	0.624	0.463	0.000
R_2 Indicator	0.152	0.235	0.152	0.000

4D Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.000	0.667	0.000
Ccomb LSpure	0.333	-	0.667	0.000
Cpure LScomb	0.333	0.000	-	0.000
Ccomb LScomb	0.667	0.500	0.833	-

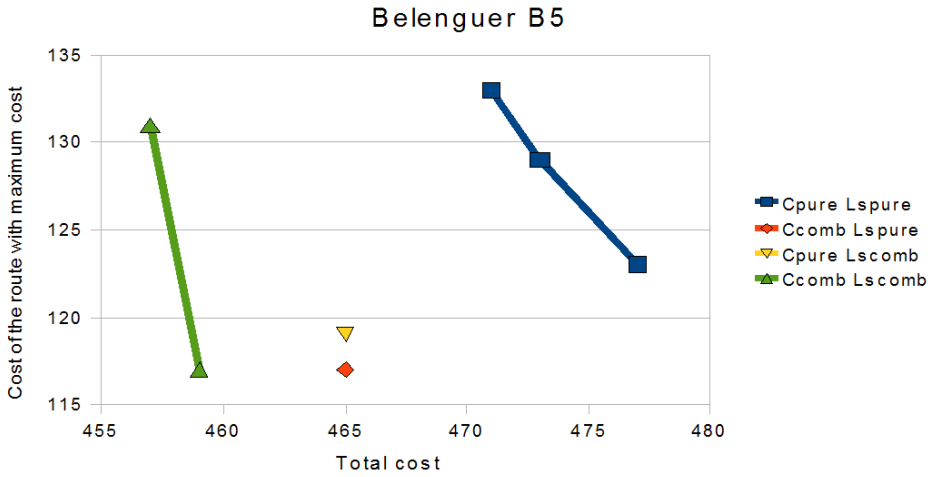
Pareto-optimal solutions on the Belenguer 5A

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
5A	427	143	425	143	423	143	423	143
	423	145	423	157				



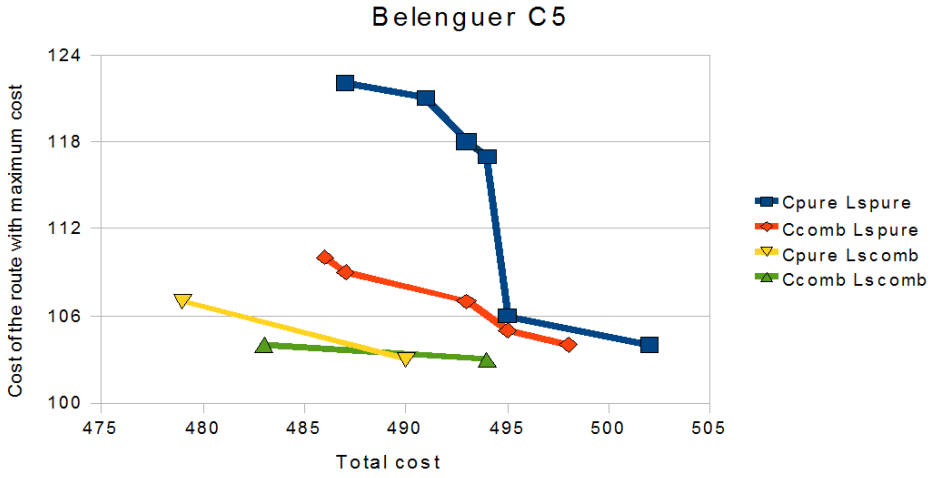
Pareto-optimal solutions on the Belenguer 5B

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
5B	477	123	465	117	465	119	459	117
	473	129					457	131
	471	133						



Pareto-optimal solutions on the Belenguer 5C

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
5C	502	104	498	104	490	103	494	103
	495	106	495	105	479	107	483	104
	494	117	493	107				
	493	118	487	109				
	491	121	486	110				
	487	122						

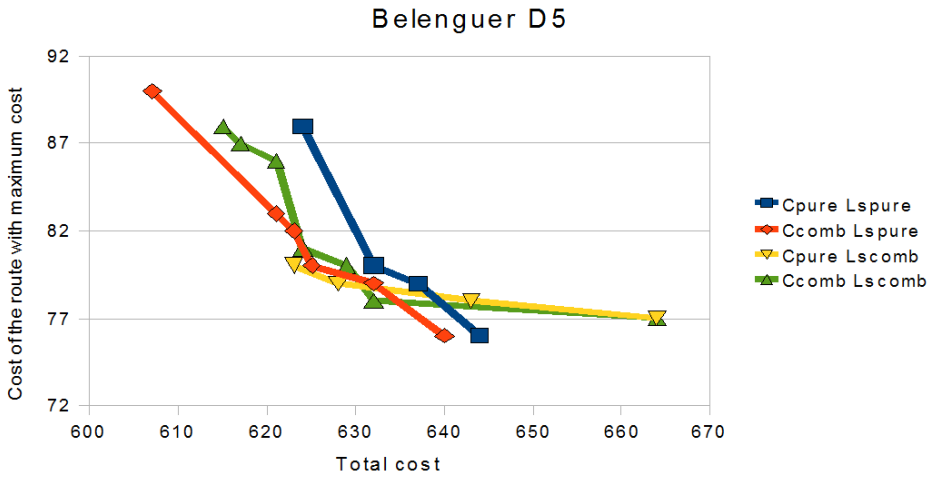


5C	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.200	0.100	0.300	0.227
Hypervolume	0.148	0.055	0.256	0.138
R_2 Indicator	0.040	0.011	0.072	0.037

5C Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.000	0.000	0.000
Ccomb LSpure	1.000	-	0.000	0.000
Cpure LScomb	1.000	1.000	-	0.500
Ccomb LScomb	1.000	1.000	0.000	-

Pareto-optimal solutions on the Belenguer 5D

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
5D	644	76	640	76	664	77	664	77
	637	79	632	79	643	78	632	78
	632	80	625	80	628	79	629	80
	624	88	623	82	623	80	624	81
			621	83			621	86
			607	90			617	87
							615	88



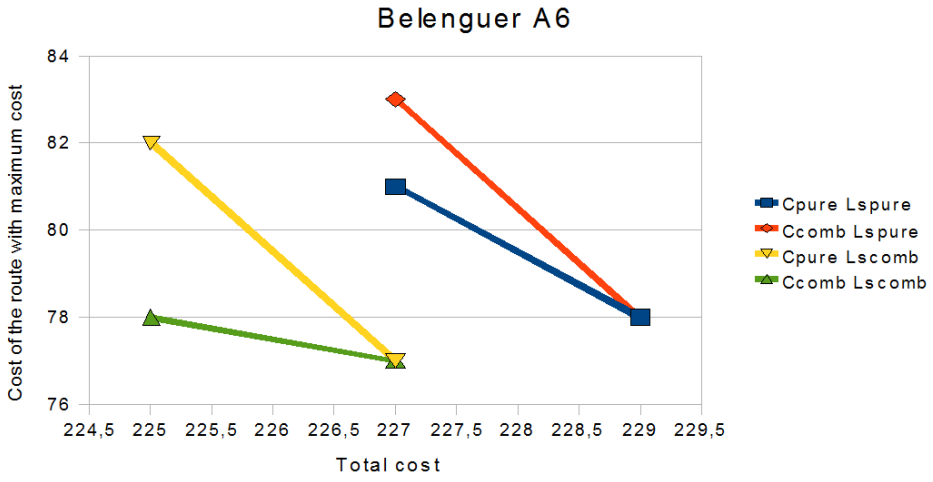
5D	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.609	0.304	0.158	0.174
Hypervolume	0.706	0.427	0.048	0.164
R_2 Indicator	0.201	0.135	0.021	0.042

5D Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.000	0.250	0.143
Ccomb LSpure	1.000	-	0.500	0.429
Cpure LScomb	0.750	0.500	-	0.286
Ccomb LScomb	0.750	0.167	0.250	-

Pareto-optimal solutions on the Belenguer 6A

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
6A	229	78	229	78	225	82	225	78
	227	81	227	83	227	77	227	77

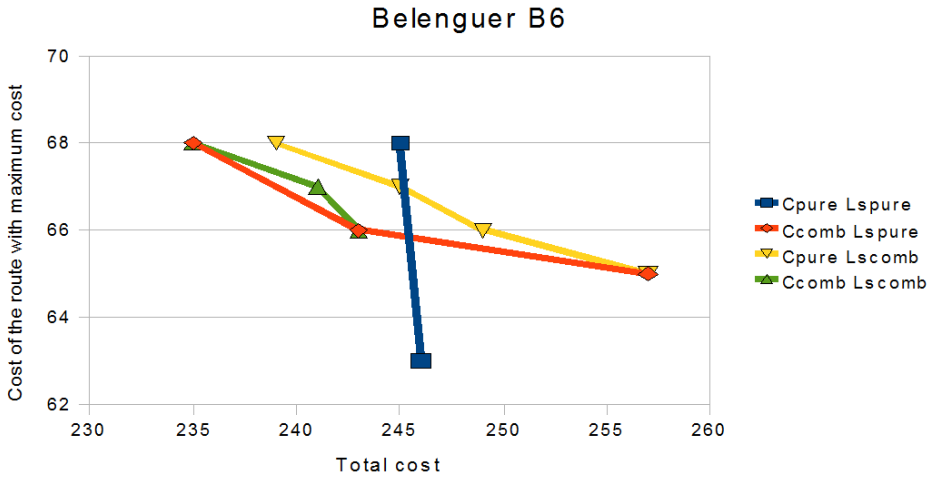
6A	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.298	0.105	0.281	0.143
Hypervolume	0.196	0.040	0.147	0.108
R_2 Indicator	0.074	0.009	0.051	0.027



6A Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.500	0.000	0.000
Ccomb LSpure	0.000	-	0.000	0.000
Cpure LScomb	1.000	1.000	-	0.000
Ccomb LScomb	1.000	1.000	0.500	-

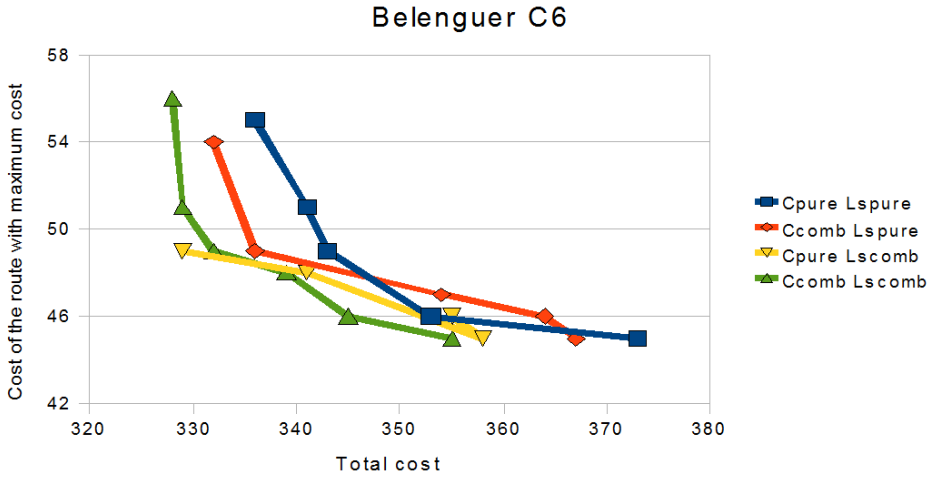
Pareto-optimal solutions on the Belenguer 6B

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
6B	245	68	257	65	257	65	235	68
	246	63	243	66	249	66	241	67
			235	68	245	67	243	66
					239	68		



Pareto-optimal solutions on the Belenguer 6C

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
6C	373	45	367	45	355	46	355	45
	353	46	364	46	358	45	345	46
	343	49	354	47	341	48	339	48
	341	51	336	49	329	49	332	49
	336	55	332	54			329	51
							328	56

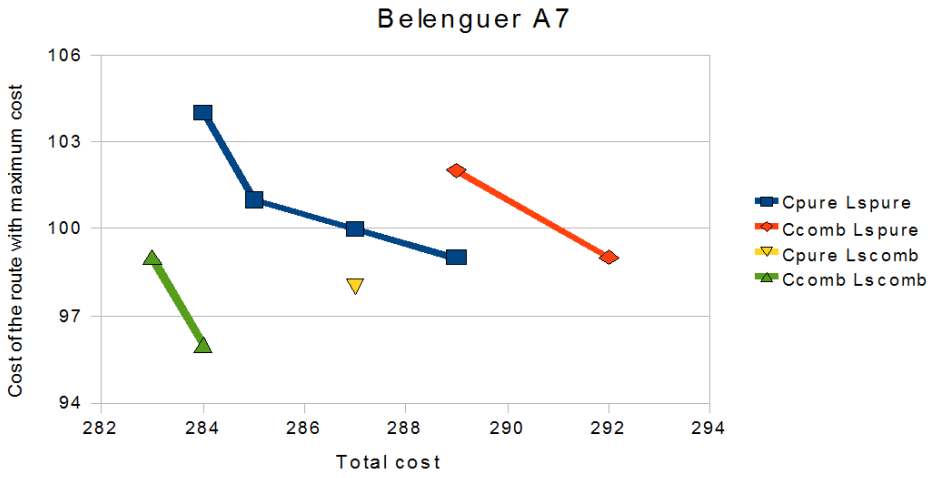


6C	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.500	0.833	0.500	0.000
Hypervolume	0.817	0.983	0.333	0.000
R_2 Indicator	0.258	0.284	0.092	0.000

6C Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.400	0.250	0.000
Ccomb LSpure	0.800	-	0.000	0.000
Cpure LScomb	0.800	0.800	-	0.333
Ccomb LScomb	1.000	1.000	0.750	-

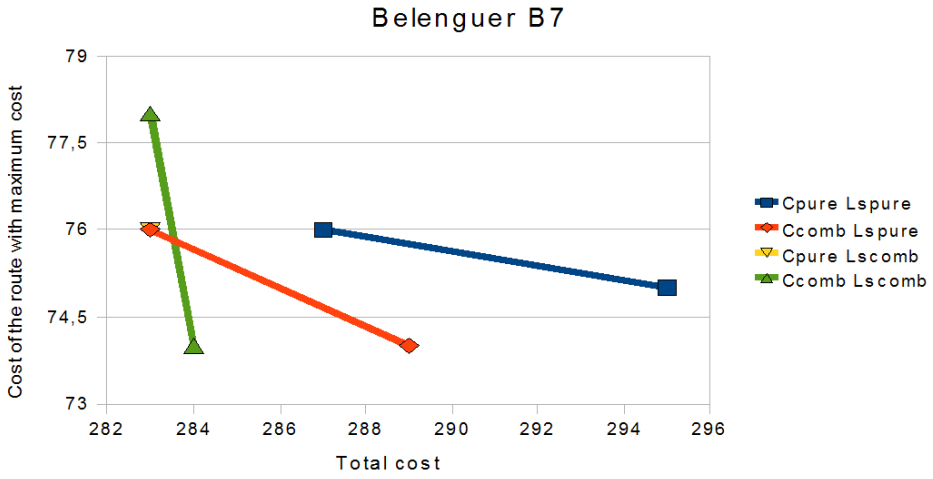
Pareto-optimal solutions on the Belenguer 7A

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
7A	289	99	292	99	287	98	284	96
	287	100	289	102			283	99
	285	101						
	284	104						



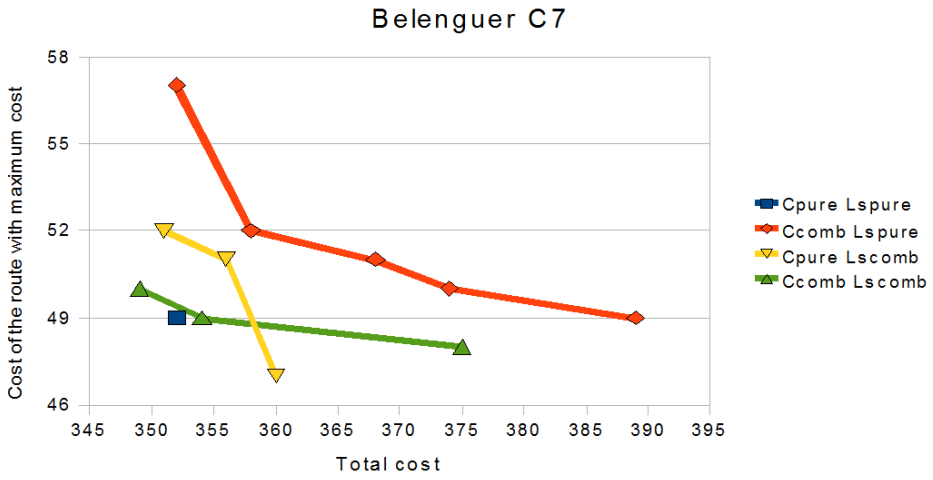
Pareto-optimal solutions on the Belenguer 7B

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
7B	295	75	289	74	283	76	284	74
	287	76	283	76			283	78



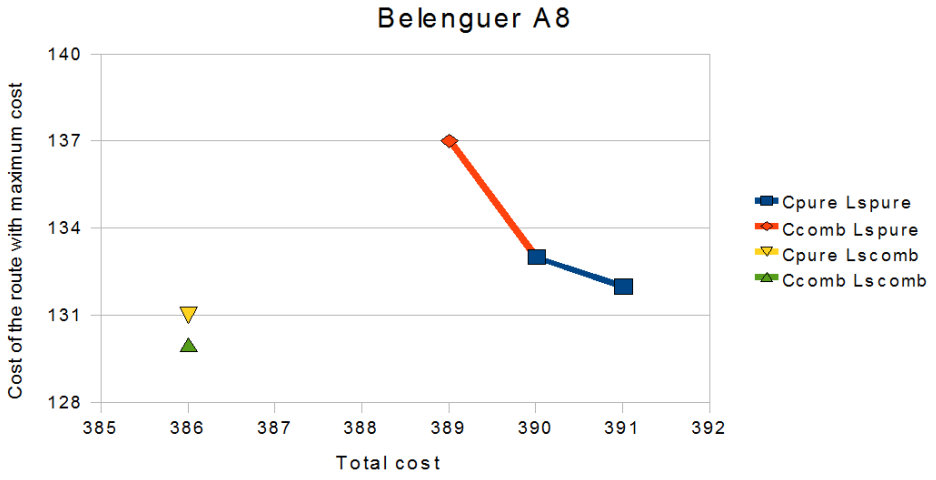
Pareto-optimal solutions on the Belenguer 7C

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
7C	352	49	389	49	360	47	375	48
			374	50	356	51	354	49
			368	51	351	52	349	50
			358	52				
			352	57				



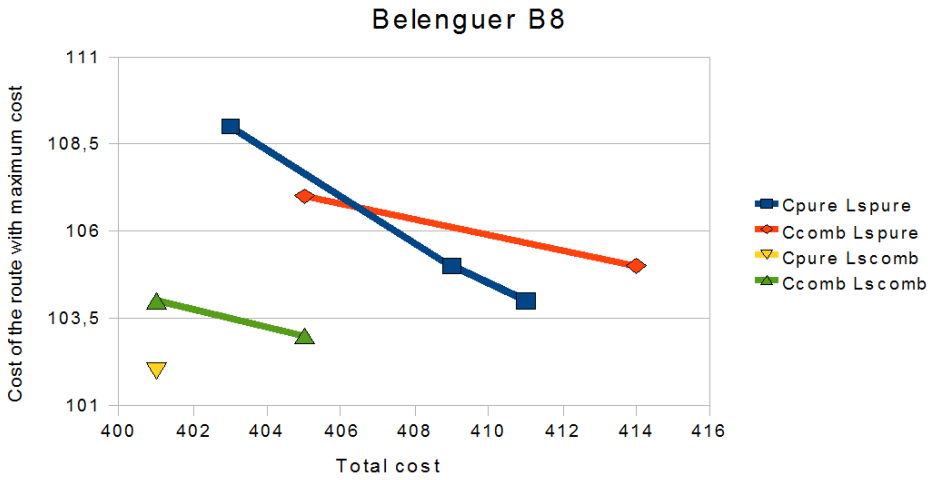
Pareto-optimal solutions on the Belenguer 8A

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
8A	391	132	390	133	386	131	386	130
	390	133	389	137				



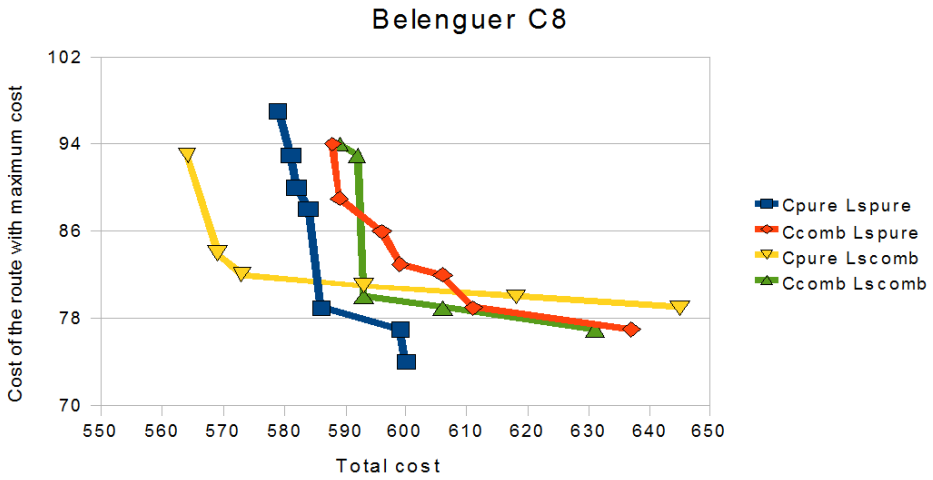
Pareto-optimal solutions on the Belenguer 8B

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
8B	411	104	414	105	401	102	405	103
	409	105	405	107			401	104
	403	109						



Pareto-optimal solutions on the Belenguer 8C

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
8C	600	74	637	77	645	79	631	77
	599	77	611	79	618	80	606	79
	586	79	606	82	593	81	593	80
	584	88	599	83	573	82	592	93
	582	90	596	86	569	84	589	94
	581	93	589	89	564	93		
	579	97	588	94				

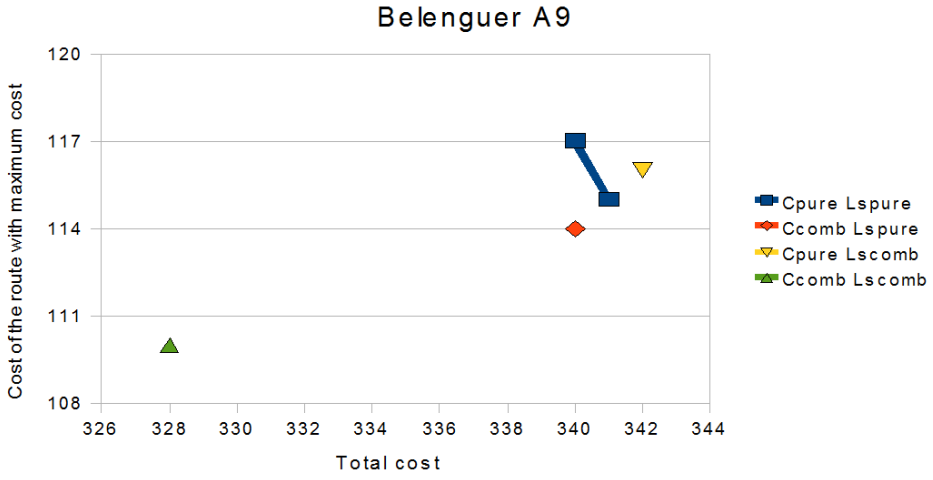


8C	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.267	0.200	0.182	0.067
Hypervolume	0.282	0.203	0.053	0.012
R_2 Indicator	0.069	0.044	0.014	0.003

8C Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	1.000	0.500	1.000
Ccomb LSpure	0.000	-	0.333	0.400
Cpure LScomb	0.571	0.714	-	0.400
Ccomb LScomb	0.000	0.714	0.500	-

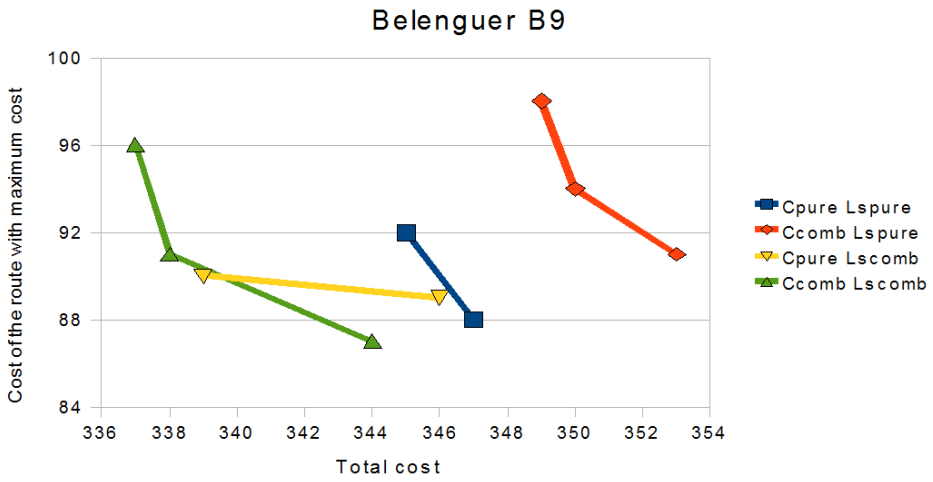
Pareto-optimal solutions on the Belenguer 9A

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
9A	341	115	340	114	342	116	328	110
	340	117						



Pareto-optimal solutions on the Belenguer 9B

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
9B	347	88	353	91	346	89	344	87
	345	92	350	94	339	90	338	91
			349	98			337	96

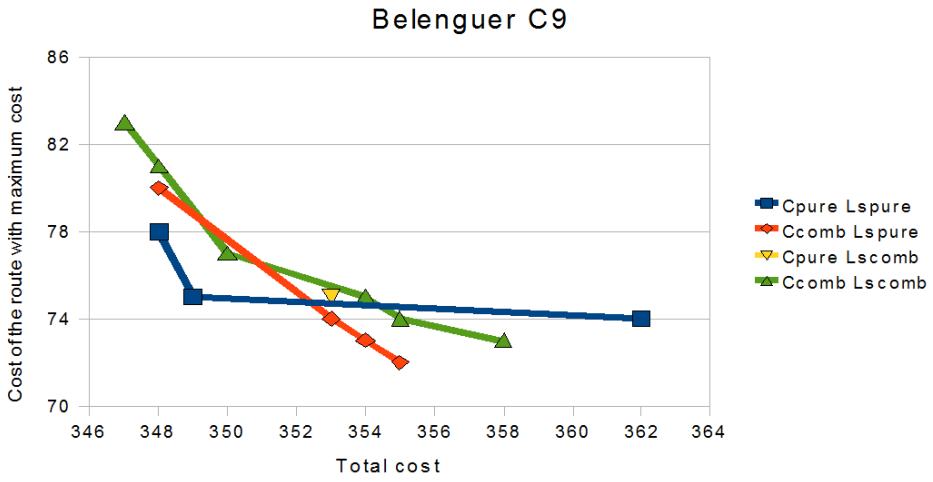


9B	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.185	0.296	0.261	0.309
Hypervolume	0.143	0.399	0.196	0.362
R_2 Indicator	0.039	0.116	0.053	0.107

9B Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	1.000	0.000	0.000
Ccomb LSpure	0.000	-	0.000	0.000
Cpure LScomb	0.500	1.000	-	0.000
Ccomb LScomb	1.000	1.000	0.500	-

Pareto-optimal solutions on the Belenguer 9C

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
9C	362	74	355	72	353	75	358	73
	349	75	354	73			355	74
	348	78	353	74			354	75
			348	80			350	77
							348	81
							347	83

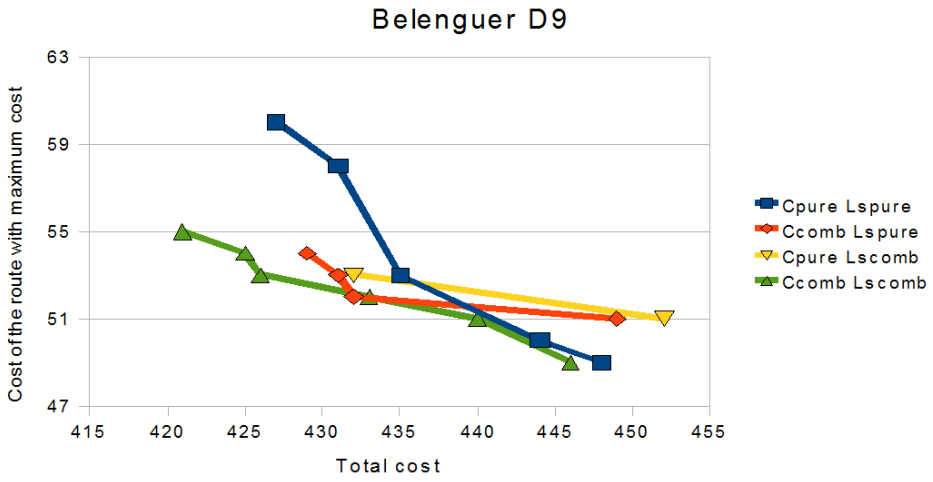


9C	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.500	0.750	0.182	0.091
Hypervolume	0.491	0.884	0.195	0.028
R_2 Indicator	0.180	0.361	0.047	0.011

9C Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.250	0.250	0.500
Ccomb LSpure	0.333	-	0.000	0.667
Cpure LScomb	0.333	0.000	-	0.667
Ccomb LScomb	0.333	0.000	0.000	-

Pareto-optimal solutions on the Belenguer 9D

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
9D	448	49	449	51	452	51	446	49
	444	50	432	52	432	53	440	51
	435	53	431	53			433	52
	431	58	429	54			426	53
	427	60					425	54
							421	55

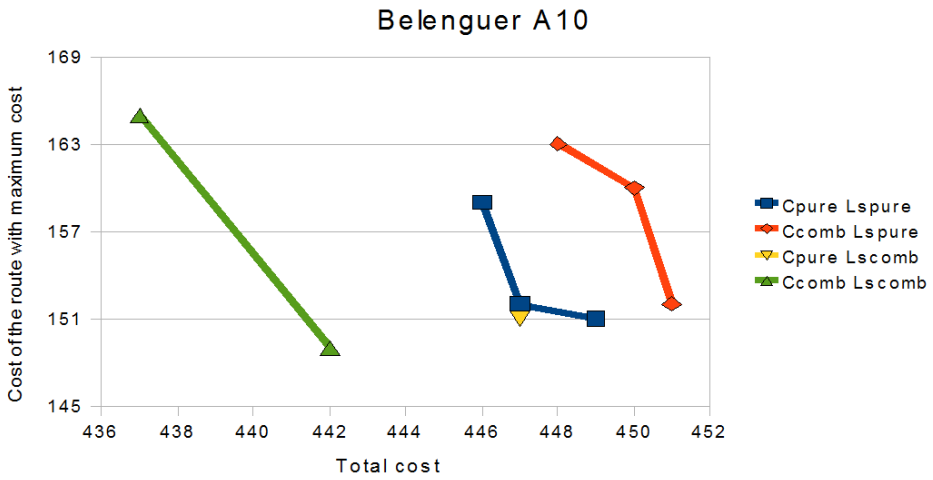


9D	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.273	0.267	0.267	0.182
Hypervolume	0.170	0.140	0.140	0.191
R_2 Indicator	0.041	0.040	0.040	0.046

9D Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.250	0.000	0.000
Ccomb LSpure	0.400	-	0.400	0.167
Cpure LScomb	0.000	0.250	-	0.000
Ccomb LScomb	0.800	0.750	0.800	-

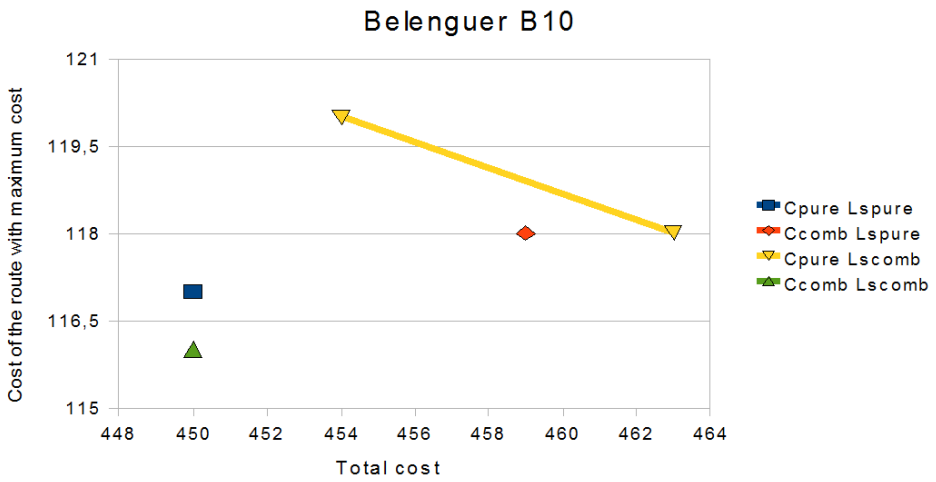
Pareto-optimal solutions on the Belenguer 10A

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
10A	449	151	451	152	447	151	442	149
	447	152	450	160			437	165
	446	159	448	163				



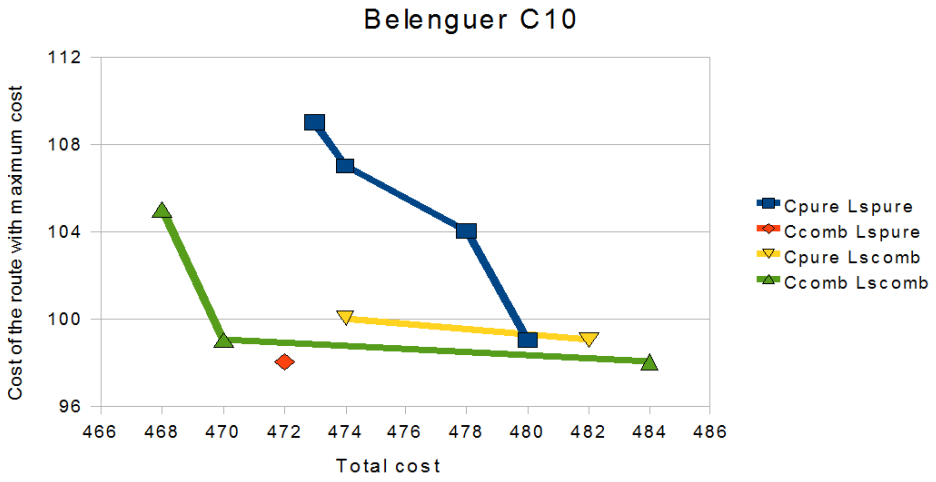
Pareto-optimal solutions on the Belenguer 10B

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
10B	450	117	459	118	463	118	450	116
					454	120		



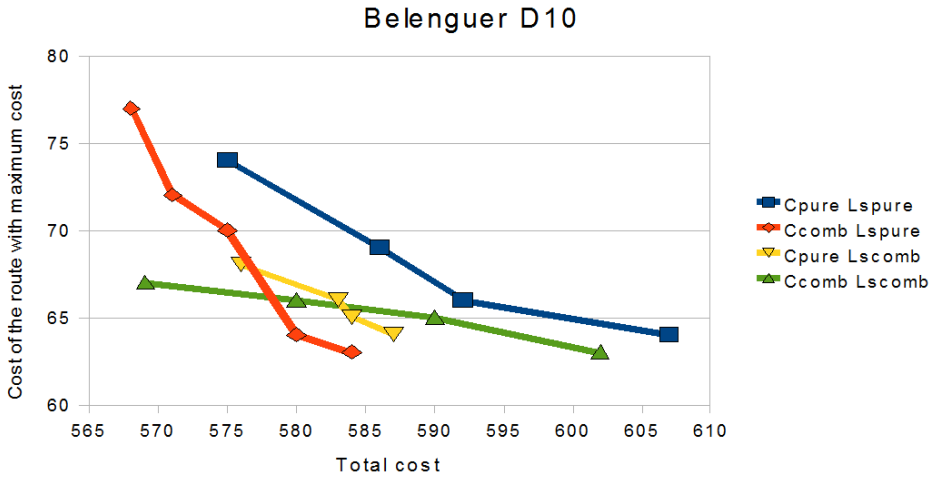
Pareto-optimal solutions on the Belenguer 10C

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
10C	480	99	472	98	482	99	484	98
	478	104			474	100	470	99
	474	107					468	105
	473	109						



Pareto-optimal solutions on the Belenguer 10D

	Cpure Lspure		Ccomb Lspure		Cpure LScomb		Ccomb LScomb	
	f_1	f_2	f_1	f_2	f_1	f_2	f_1	f_2
10D	607	64	584	63	587	64	602	63
	592	66	580	64	584	65	590	65
	586	69	575	70	583	66	580	66
	575	74	571	72	576	68	569	67
			568	77				



10D	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Epsilon Indicator	0.357	0.286	0.357	0.071
Hypervolume	0.343	0.261	0.343	0.010
R_2 Indicator	0.083	0.082	0.083	0.002

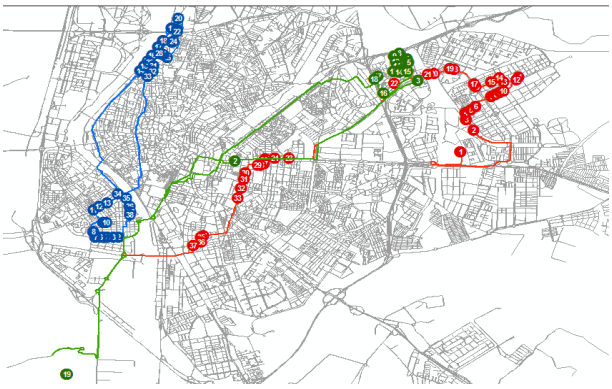
10D Coverage	Cpure LSpure	Ccomb LSpure	Cpure LScomb	Ccomb LScomb
Cpure LSpure	-	0.000	0.000	0.000
Ccomb LSpure	1.000	-	0.750	0.750
Cpure LScomb	0.750	0.000	-	0.250
Ccomb LScomb	1.000	0.400	0.500	-

Appendix B

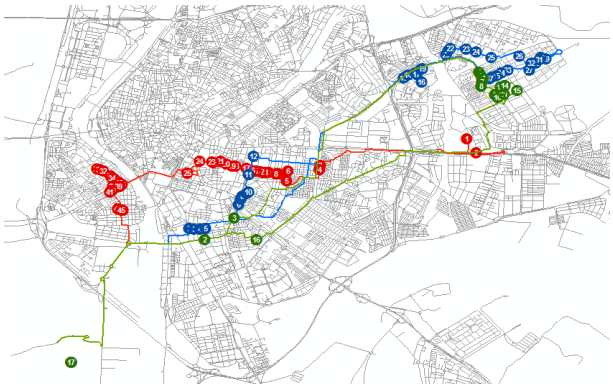
Single-Objective Waste Collection Solution

B.1 Solution for 19 routes

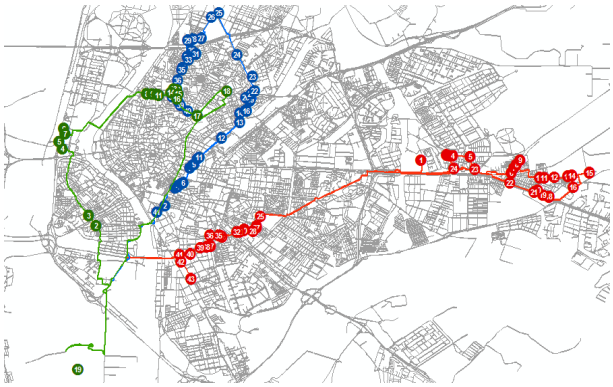
Route 1



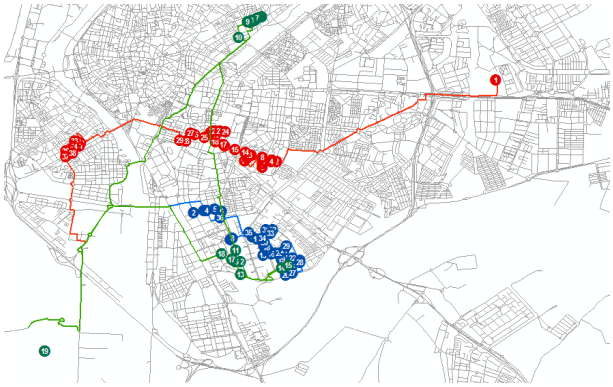
Route 2



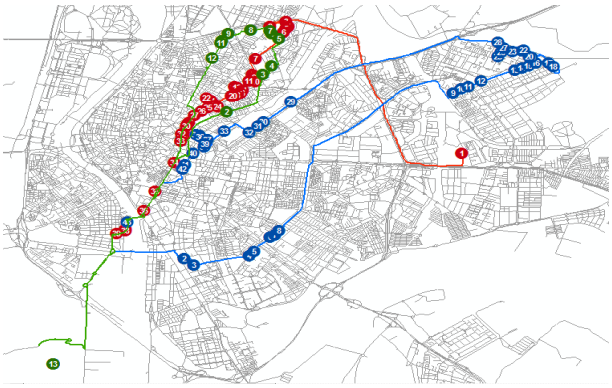
Route 3



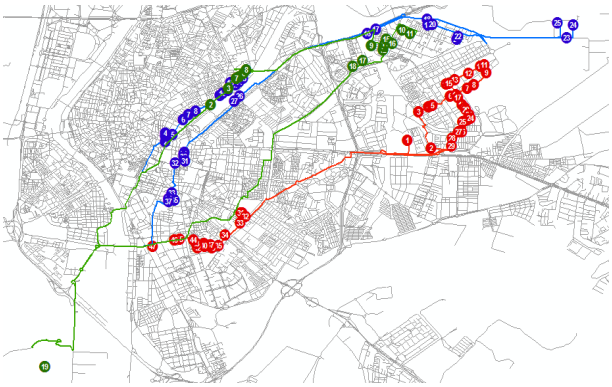
Route 4



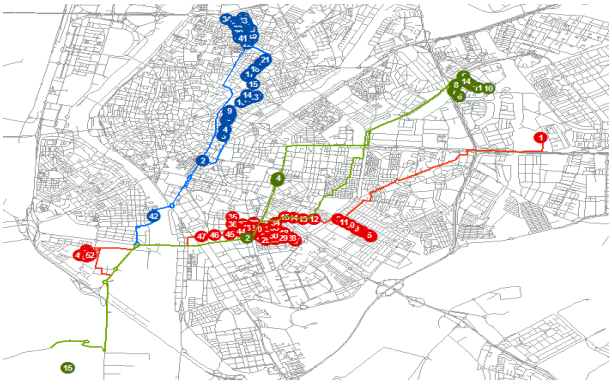
Route 5



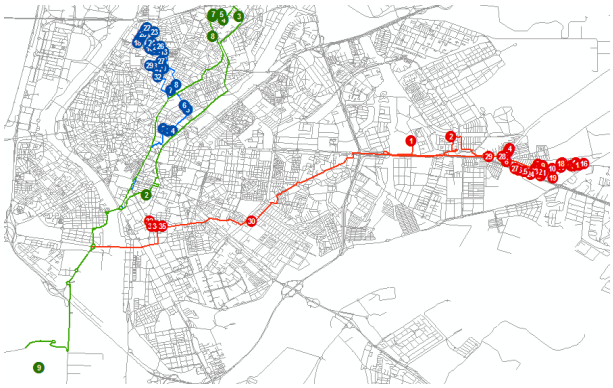
Route 6



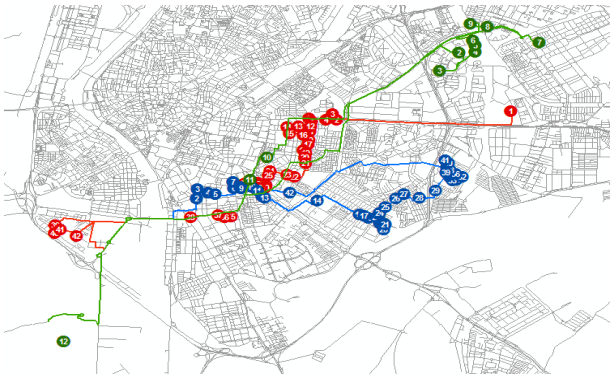
Route 7



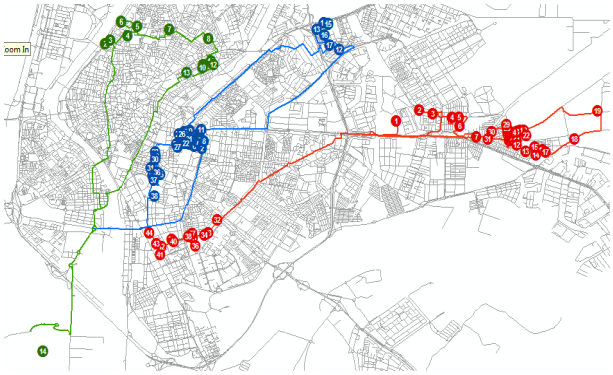
Route 8



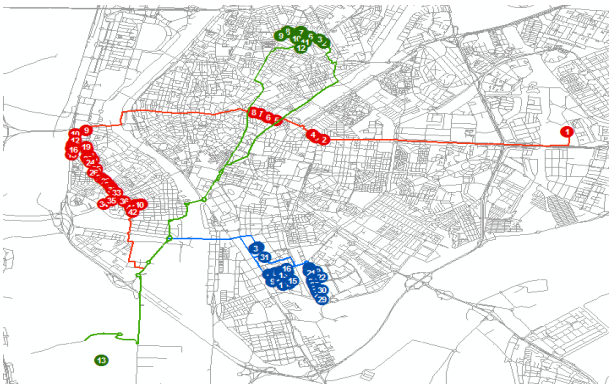
Route 9



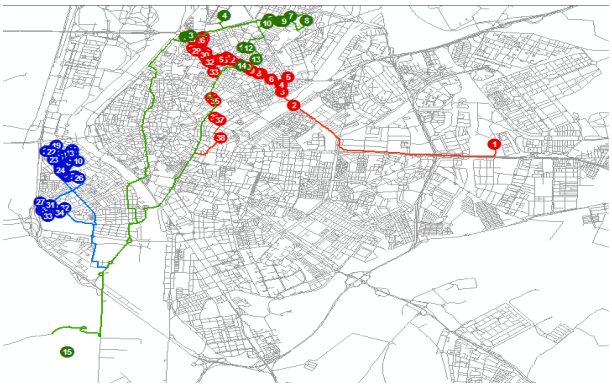
Route 10



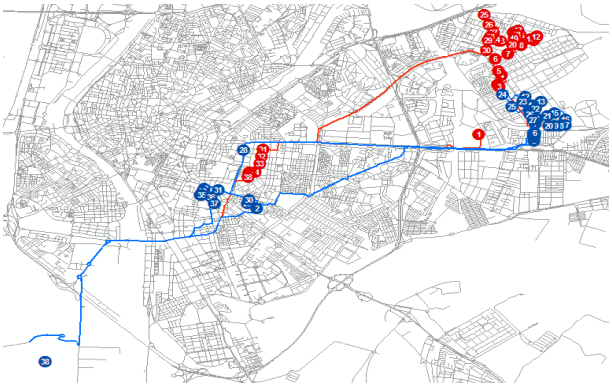
Route 11



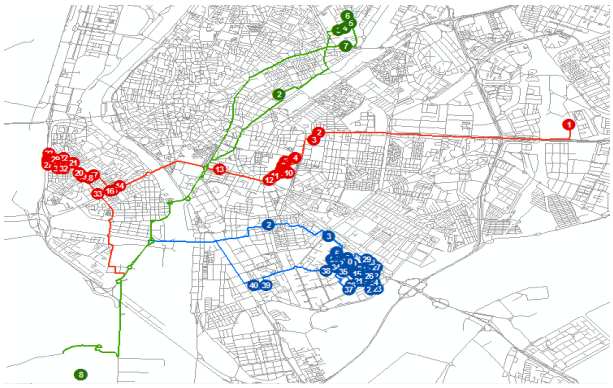
Route 12



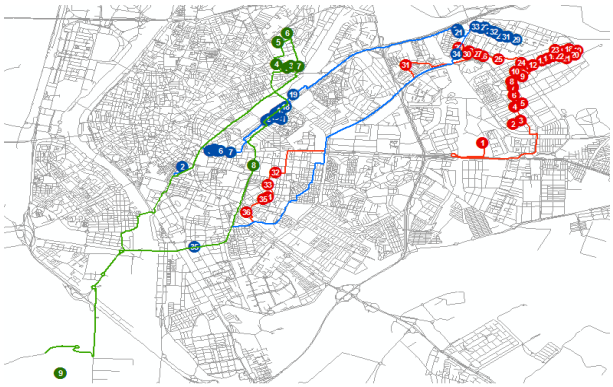
Route 13



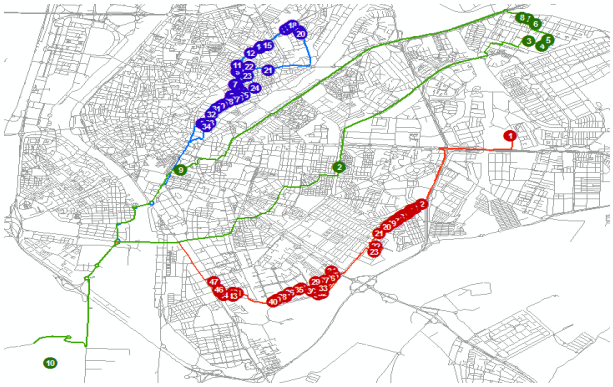
Route 14



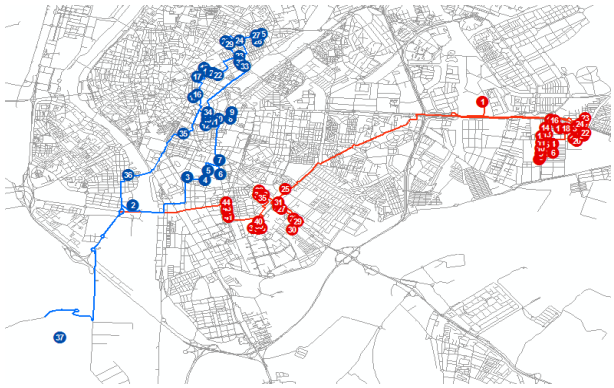
Route 15



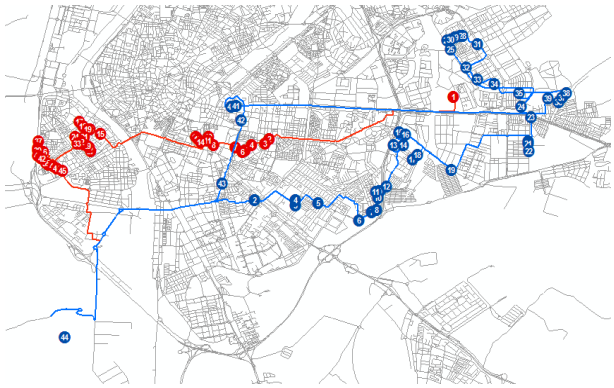
Route 16



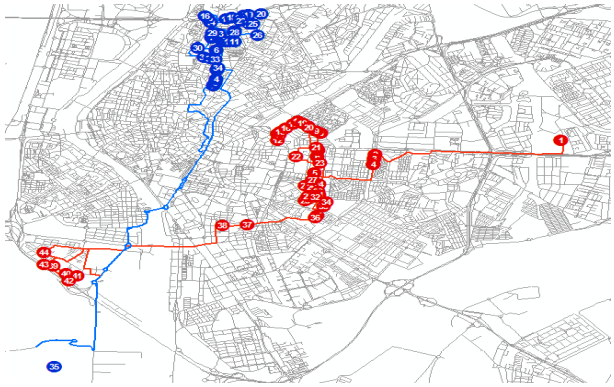
Route 17



Route 18

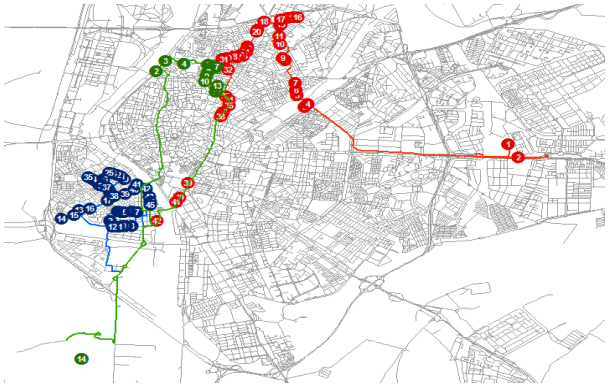


Route 19

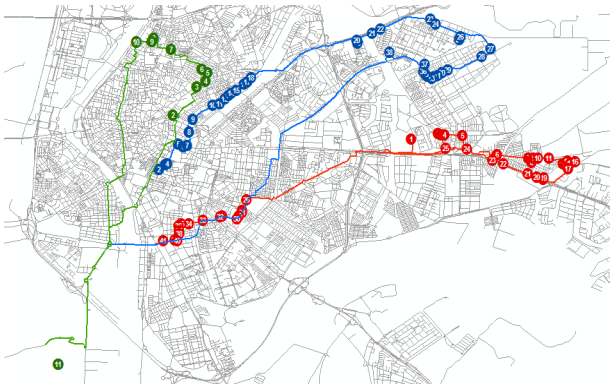


B.2 Solution for 20 routes

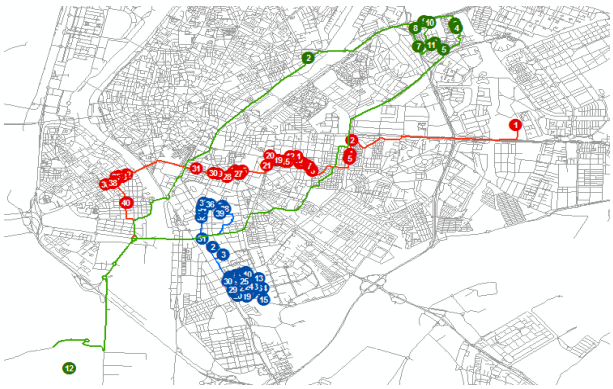
Route 1



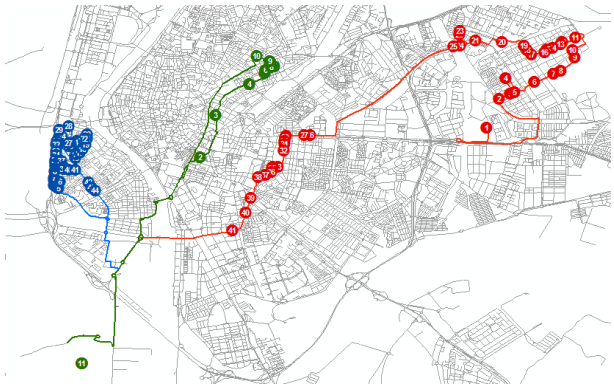
Route 2



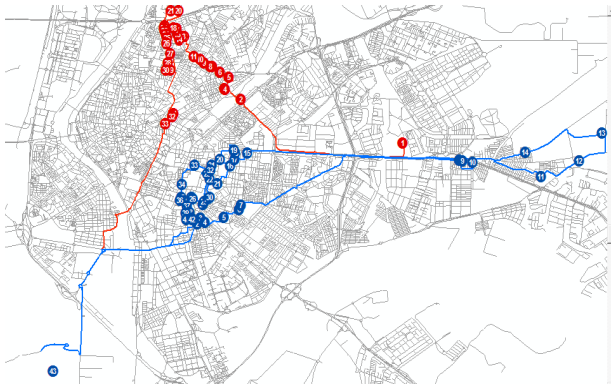
Route 3



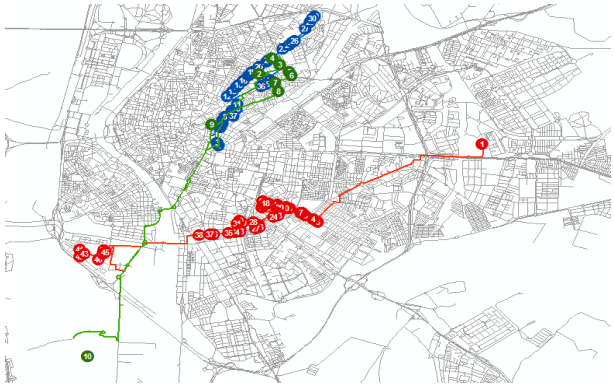
Route 4



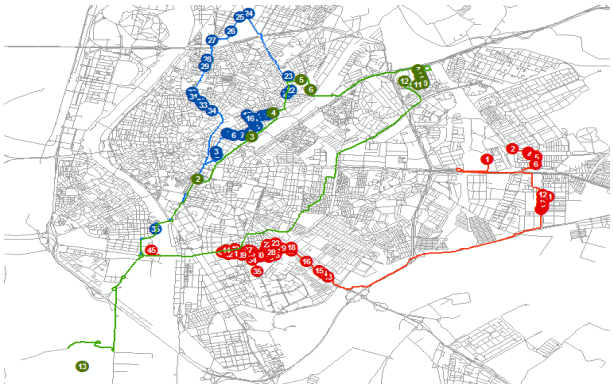
Route 5



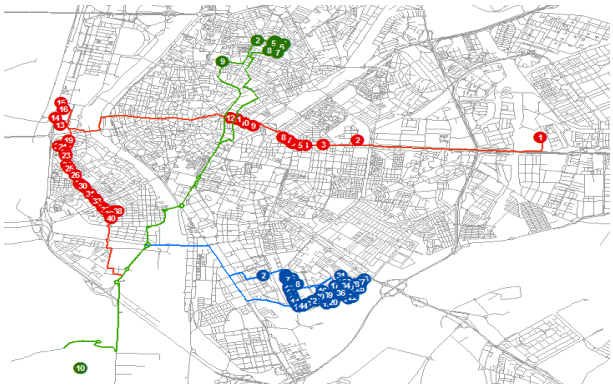
Route 6



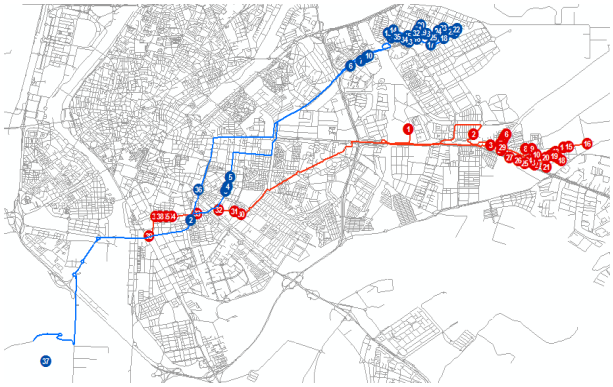
Route 7



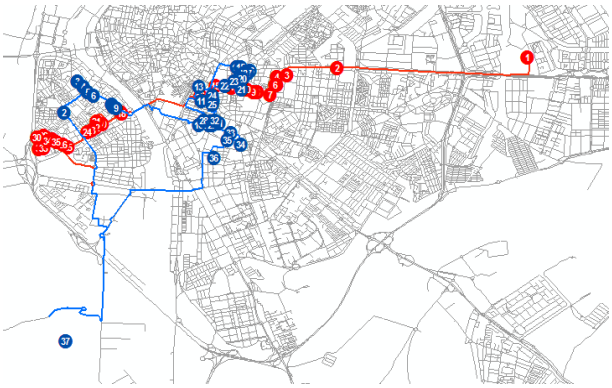
Route 8



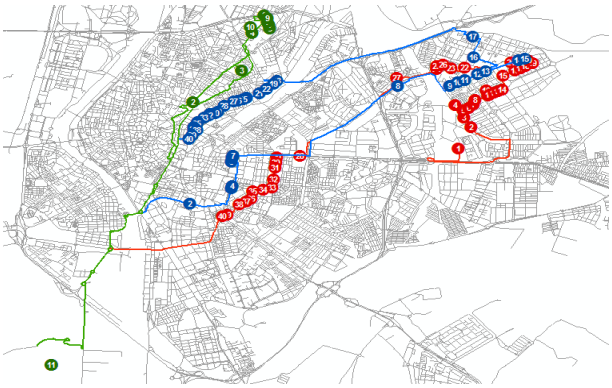
Route 9



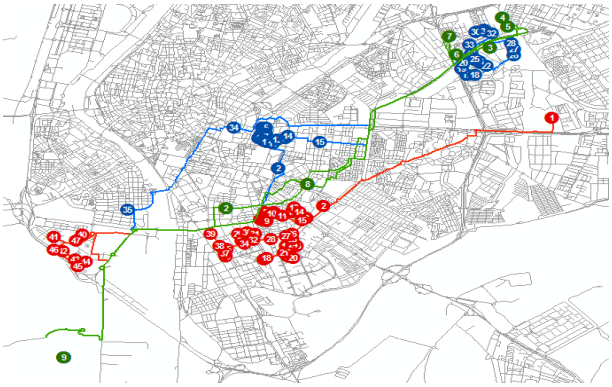
Route 10



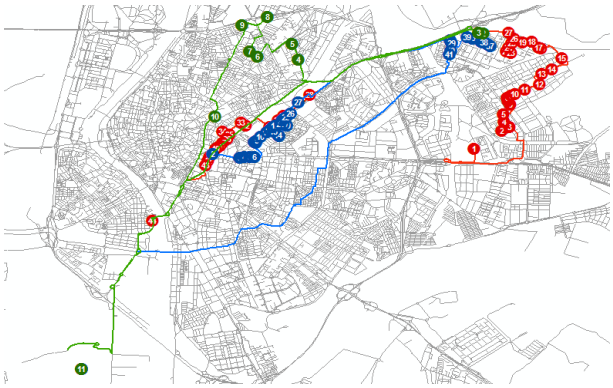
Route 11



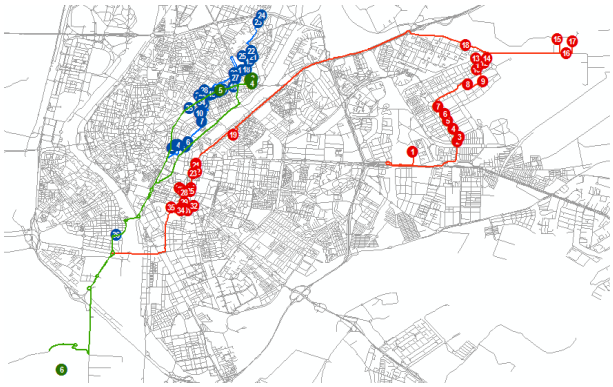
Route 12



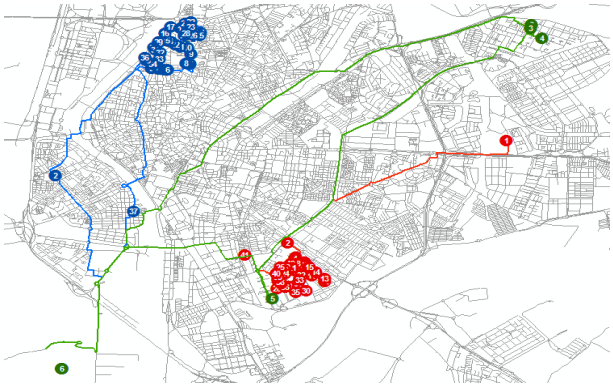
Route 13



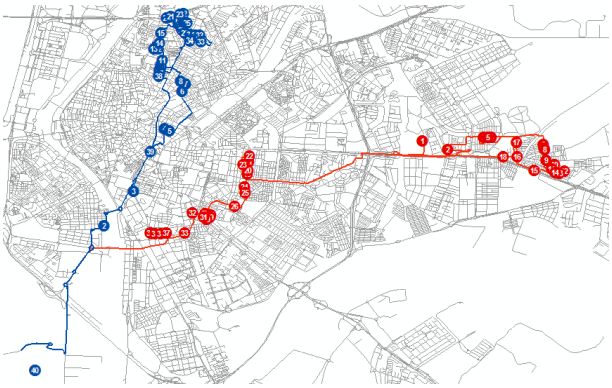
Route 14



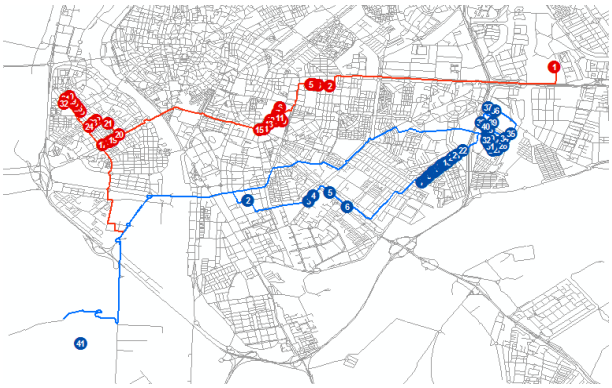
Route 15



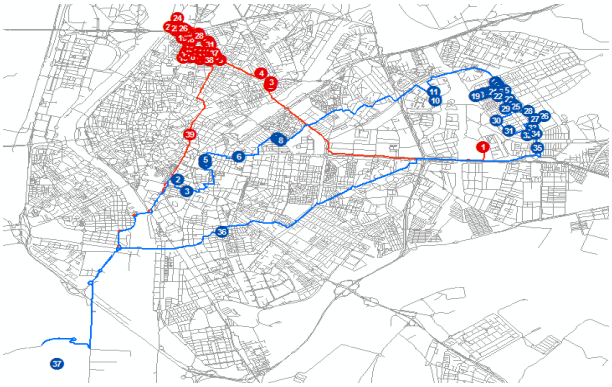
Route 16



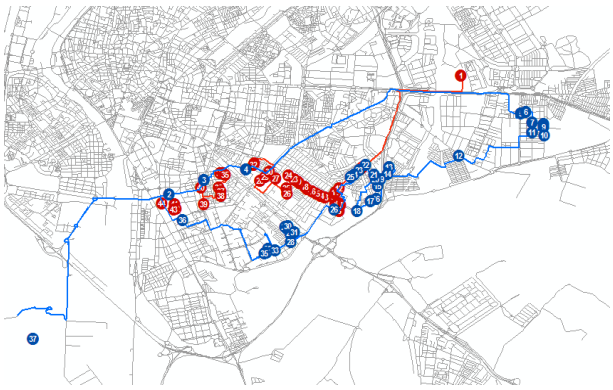
Route 17



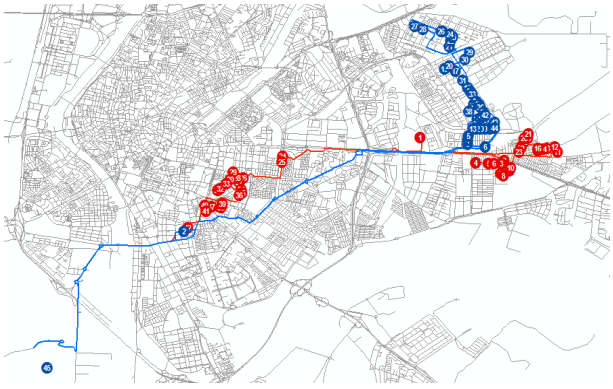
Route 18



Route 19

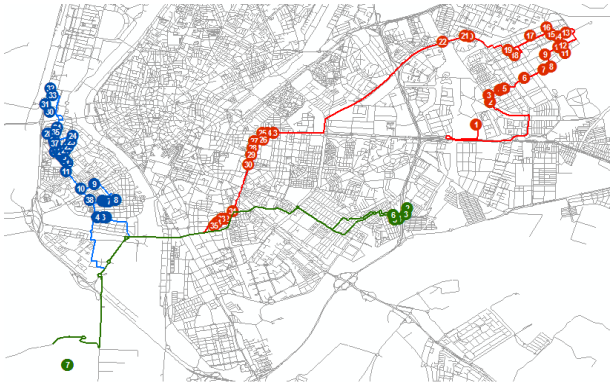


Route 20

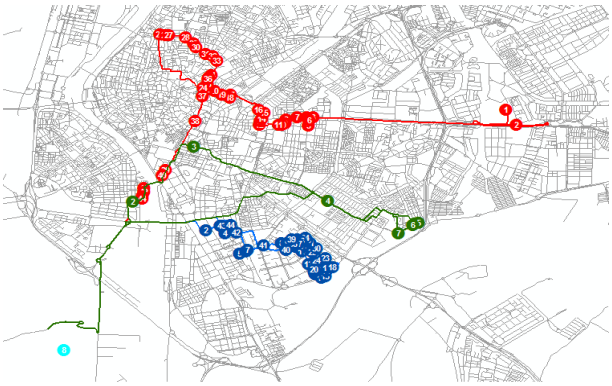


B.3 Solution for 21 routes

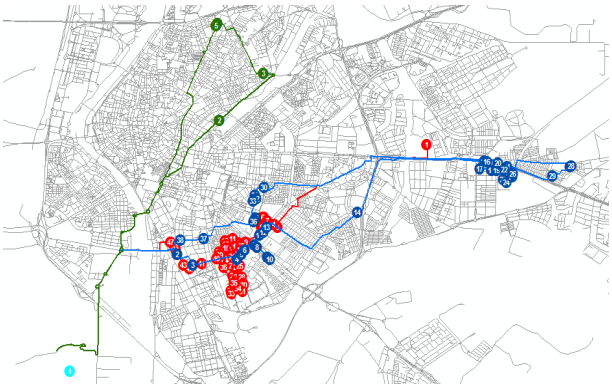
Route 1



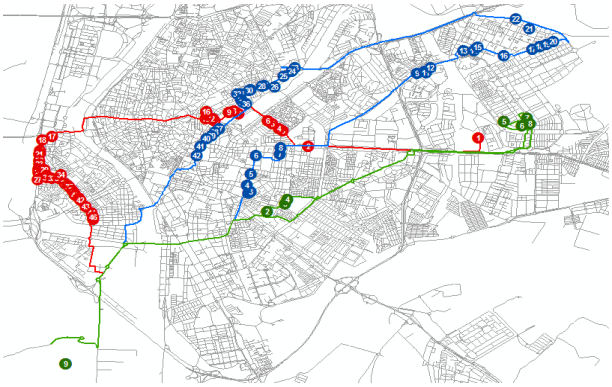
Route 2



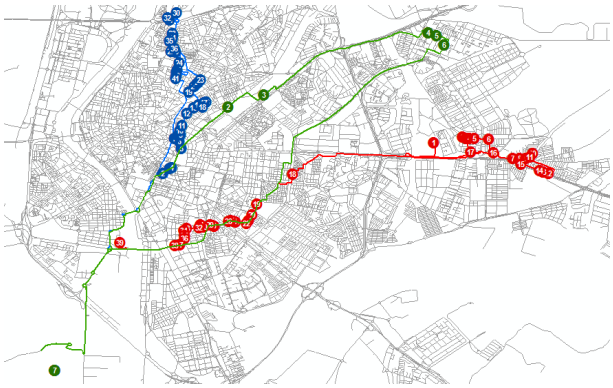
Route 3



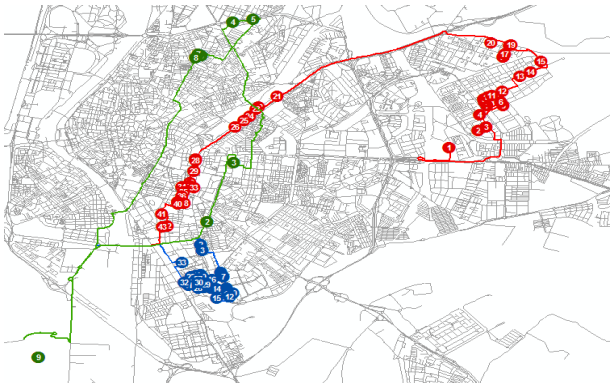
Route 4



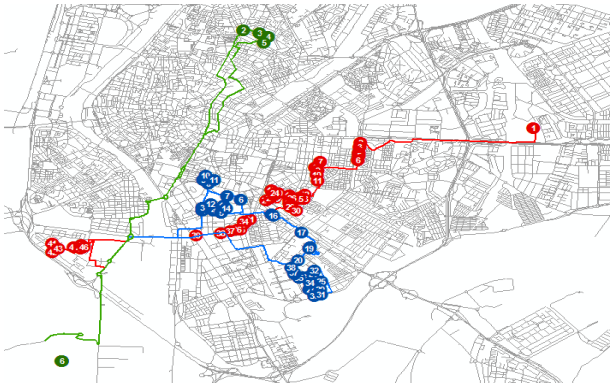
Route 5



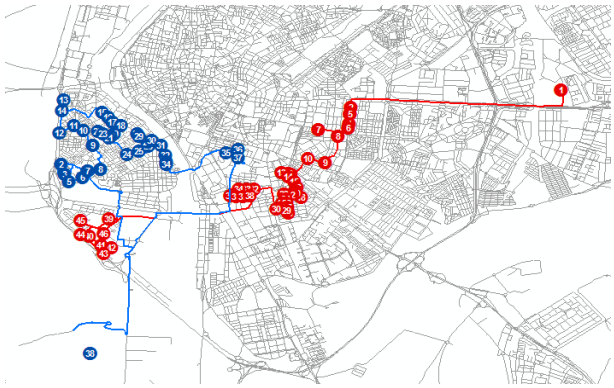
Route 6



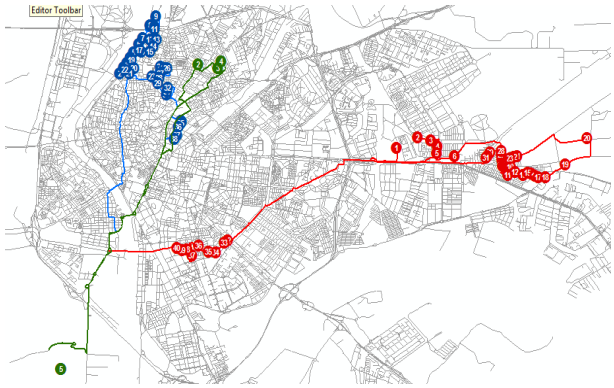
Route 7



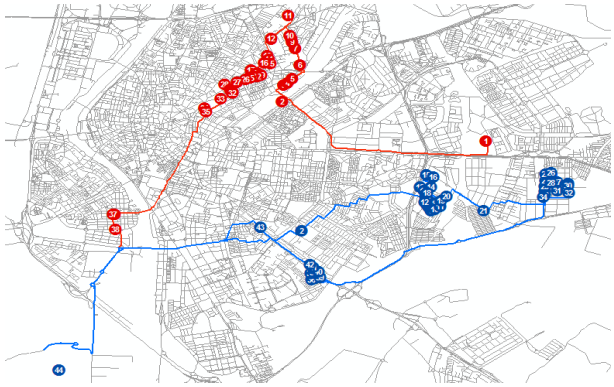
Route 8



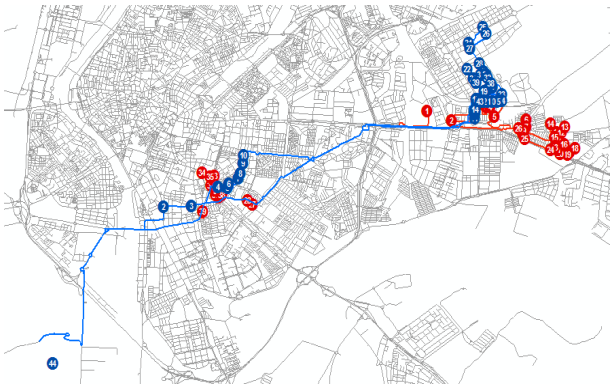
Route 9



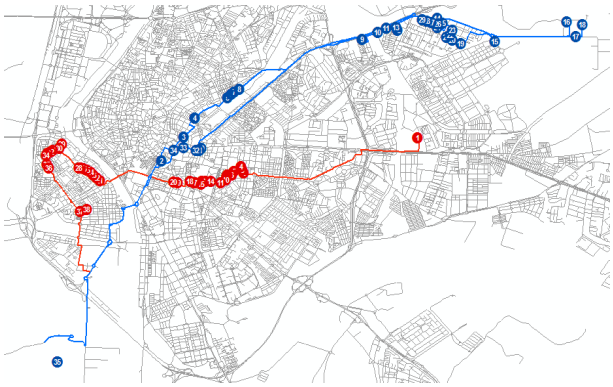
Route 10



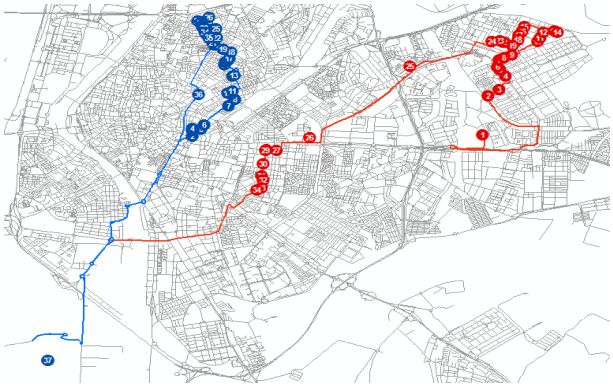
Route 11



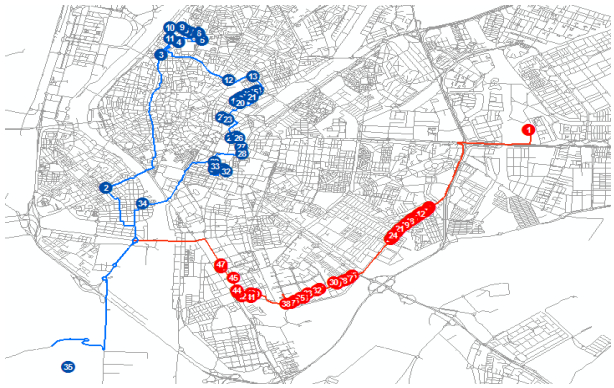
Route 12



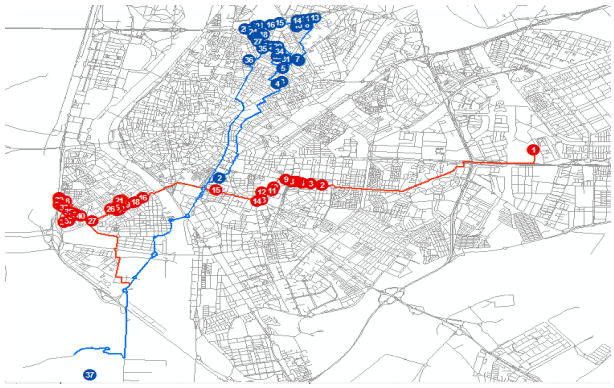
Route 13



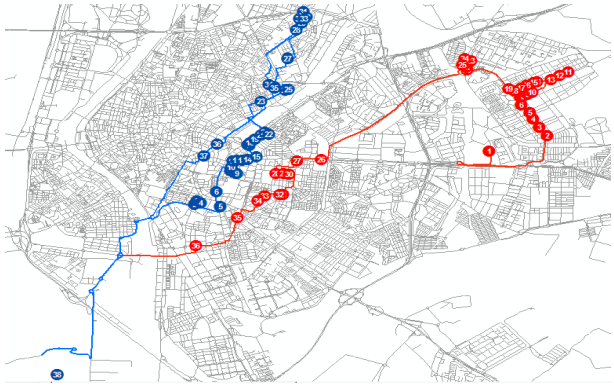
Route 14



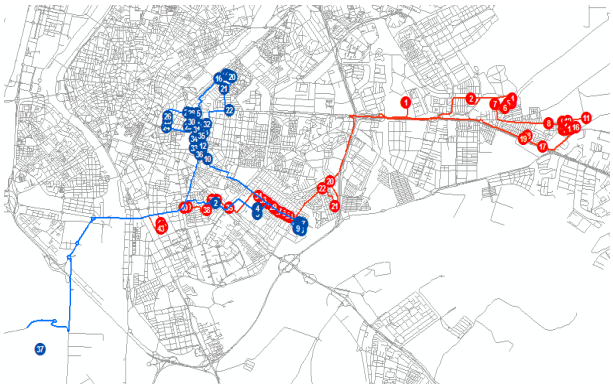
Route 15



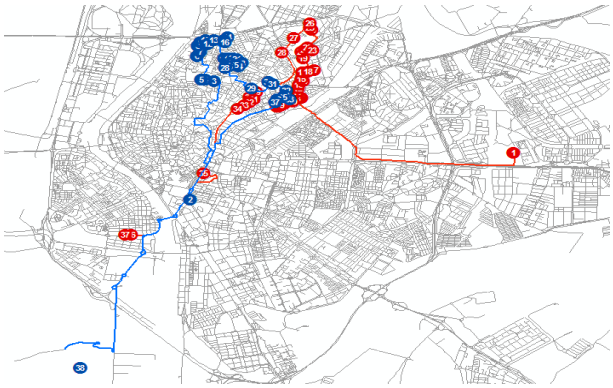
Route 16



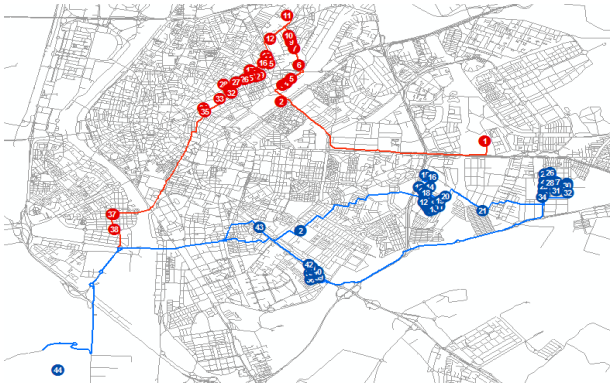
Route 17



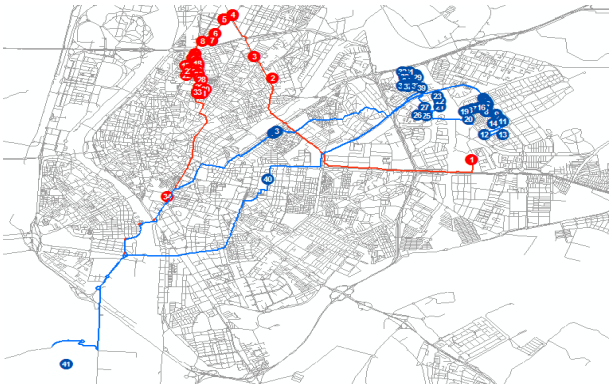
Route 18



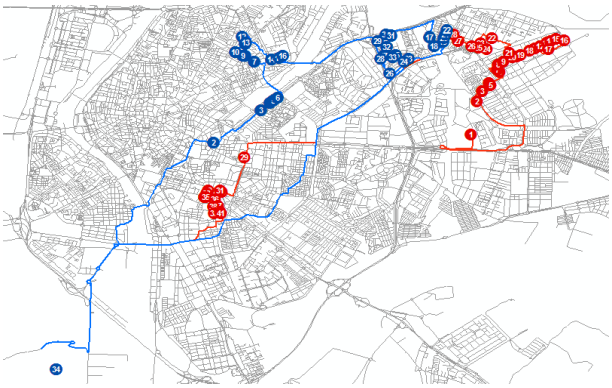
Route 19



Route 20



Route 21



Bibliography

[1] Araque J., Kudva G., Morin T. and Pekny J. (1994). A branch-and-cut algorithm for the vehicle routing problem. *Annals of Operations Research,* **50**, 37–59.

[2] Assad A. (1988). Modeling and Implementation Issues in Vehicle Routing. *Vehicle Routing: Methods and Studies,* 7–45.

[3] Assad A.A. and Golden B.L. (1995). Arc routing methods and applications. *Handbookd in operations research and management science - Network rounting,* **8**, 375–483.

[4] Baker B.M. and Ayechew M.A. (2003). A genetic algorithm for the vehicle routing problem. *Computers & Operations Research,* **30**, 787–800.

[5] Baker B.M. and Carreto C.A.C. (2003). A visual interactive approach to vehicle routing. *Computers & Operations Research,* **30**, 321–337.

[6] Baldacci R, Hadjiconstantinou E. and Mingozzi A. (2004). An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research,* **52**, 5, 723–738.

[7] Baldacci R. and Maniezzo V. (2006). Exact Methods Based on Node-Routing Formulations for Undirected Arc Routing Problems. *Networks* **47**, 3, 52–60.

[8] Balinsky M. and Quandt R. (1964). On an integer program for a delivery problem. *Operations Research,* **12**, 300–304.

[9] Baràn B. and Schaerer M. (2003). A multiobjective ant colony system for vehicle routing problem with time windows. *Proceedings of the Twenty-first IASTED International Conference on Applied Informatics,* 97–102.

[10] Bautista J., Fernández E. and Pereira J. (2008). Solving an urban waste collection problem using ants heuristics. *Computers & Operations Research,* **35**, 3, 3020–3033.

[11] Beasley J.E. (1983). Route-first cluster-second methods for vehicle routing. *Omega,* **11**, 403–408.

[12] Belenguer J.M. and Benavent E. (1991). Polyhedral Results on the Capacitated Arc Routing Problem. *Working Paper, Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain.*

[13] Belenguer J.M. and Benavent E. (1998). The capacitated arc routing problem. *Computational Optimization and Applications,* **10**, 2, 165–187.

[14] Belenguer J.M. and Benavent E. (2003). A cutting plane algorithm for the capacitated arc routing problem. *Computers & Operations Research,* **30**, 5, 705–728.

[15] Bell J.E. and McMullen P.R. (2004). Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics,* **18** 41-48.

[16] Beltrami E. and Bodin L. (1974). Networks and vehicle routing for municipal waste collection. *Networks,* **4**, 1, 65-94.

[17] Biggs N.L., LLoyd E.K. and Wilson R.J. (1976). Graph Theory 1736–1936 *Clarendon Press, Oxford*.

[18] Blum C., Blesa-Aguilera M.J., Roli A. and Sampels M., editors. (2008). Hybrid Metaheuristics, An Emerging Approach to Optimization. *Studies in Computational Intelligence, Springer.*

[19] Bodin L., Golden B., Assad A. and Ball M. (1983). Routing and sheduling of vehicles and crews: the state of the art. *Computer & Operations Research,* **10**, 2, 63–211.

[20] Bodin L., Mingozzi A., Baldacci R. and Ball M. (2000). The Rollon-Rolloff Vehicle Routing Problem. *Transportation Science,* **6**, 3, 271–288.

[21] Bowerman B., Hall B. and Calamai P. (1995). A multi-objective optimization approach to urban school bus routing: Formulation and solution method. *Transportation Research Part A,* **29**, 107–123.

[22] Bramel J. and Simchi-Levi D. (1995). A location based heuristic for general routing problems. *Operations Research,* **43**, 649–660.

[23] Brandão J. and Eglese R. (2008). A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers & Operations Research,* **35**, 1112–1126.

[24] Bräysy O., Grendreau M., Hasle G. and Løkketangen A. (2002). A survey of rich vehicle routing models and heuristic solution techniques. *Technical report, SINTEF*.

[25] Burke E.K., Kendall G., Newall J., Hart E., Ross P., Schulemburg S. (2003). Hyper-heuristics: an Emerging Direction in Modern Search Technology. *In: Glover F.W., Kochenberger G.A. (eds.), Handbook of Metaheuristics, Kluwer Academic Publishers.*

[26] Castri-Gutierrez J., Landa-Silva D. and Moreno-Pérez J. (2011). Nature of real-world multi-objective vehicle routing with evolutionary algorithms. *IEEE Congress on Evolutionary Computation,* 257–264.

[27] Christofides N. (1973). The optimum traversal of a graph. *Omega, Elsevier,* **1**, 6, 719-732.

[28] Christofides N., Mingozzi A. and Toth P. (1979). The vehicle routing problem. In Christofides N., Mingozzi A., Toth P. and Sandi C., editors, *Combinatorial Optimization,* Wiley, Chichester, UK, 315–338.

[29] Christofides N., Mingozzi A. and Toth P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming,* **20**, 255–282.

[30] Christiansen C.H., Lysgaard J. and Wøhlk S. (2009). A Branch-and-Price Algorithm for the Capacitated Arc Routing Problem with Stochastic Demands. *Operations Research Letters,* **37**, 6, 392–398.

[31] Clarke G. and Wright J.V. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research,* **12**, 568–581.

[32] Corberán A., Fernández E., Laguna M. Martí R. (2002). Heuristic solutions to the problem of routing school buses with multiple objectives. *Journal of the Operational Research Society,* **53**, 427-435.

[33] Cornuéjols G. and Harche F. (1993). Polyhedral study of the capacitated vehicle routing problem. *Mathematical Programming,* **60**, 21–52.

[34] Dantzig G.B. and Ramser J.H. (1959). The Truck Dispatching Problem. *Management Science,* **6**, 1, 80–91.

[35] DeArmon J. (1981). A comparison of heuristics for the chinese postman problem. *MSc Thesis, University of Maryland.*

[36] Desrochers M. and Laporte G. (1991). Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. *Operations Research Letters,* **10**, 27–36.

[37] Doerner K.F. and Schmid V. (2010). Survey: Matheuristics for Rich Vehicle Routing Problems. *Hybrid Metaheuristics. Lecture Notes in Computer Science* **6625**, 206–221.

[38] Dorigo M., Maniezzo V. and Colorni A. The ant system: Ant autocatalytic optimizing process. *Technical Report TR91-016, Politenico di Milano, 1991.*

[39] Dror M. (2000). Arc Routing: Theory, Solutions and Applications. *Kluwer Academic Publishing, Boston, MA, USA.*

[40] Dror M., Stern H. and Trudeau P. (1987). Postman tour on a graph with precedence relation on arcs. *Networks,* **17**, 3, 283-294.

[41] Dror M. and Trudeau P. (1989). Savings by split delivery routing. *Transportation Science,* **23**, 141-145.

[42] Dror M. and Trudeau P. (1990). Split delivery routing. *Naval Research Logistics,* **37**, 383-402.

[43] Eilon S., Watson-Gandy C.D.T. and Christofides N. (1971). Distribution Management: Mathematical Modelling and Practical Analysis. *Griffin, London.*

[44] Eiselt H.A., Gendreau M. and Laporte G. (1995). Arc routing problems, Part 1: The chinese postman problem. *Operations Research,* **43**, 2, 231-242.

[45] Eiselt H.A., Gendreau M. and Laporte G. (1995). Arc routing problems, Part 2: The rural postman problem. *Operations Research,* **43**, 3, 399-414.

[46] El-Sherbeny N. (2001). Resolution of a vehicle routing problem with multi-objective simulated annealing method. *PhD thesis, Faculté Plytechnique de Mons, Mons, Belgique.*

[47] Euler L. (1741). Solutio Problematis ad Geometrian Situs Pertinentis. *Commentarii academiae scientarum Petropolitanae,* **8**, 128–140.

[48] Feillet D., Dejax P. and Gendreau M. (2005). The Profitable Arc Tour Problem: Solution with a Branch-and-Price Algorithm. *Transportation Science,* **39**, 539–552.

[49] Feo T.A. and Resende M.G.C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization,* **6**, 109-133.

[50] Fisher M.L. and Jaikumar R. (1981). A generalized assignment heuristic for the vehicle routing problem. *Networks,* **11**, 109–124.

[51] Finke G., Claus A. and Gunn E. (1984). A two-commodity network flow approach to the traveling salesman problem. *Congress Numer,* **41**, 167–178.

[52] Fleischmann B. (1990). The Vehicle Routing Problem with Multiple Use of vehicles. *Working paper, Fachbereich Wirts- chaftswissenschaften, Universitdt Hamburg.*

[53] Francis P. and Smilowitz K. (2006). The Period Vehicle Problem with Service Choice. *Transportation Science,* **40**(4), 439-454.

[54] Frederickson G.N., Hecht M.S. and Kim C.E. (1978). Approximation algorithms for some routing problems. *SIAM Journal of Computing,* **7**, 2, 178-193.

[55] Fukasawa R., Longo H. and Lysgaard J., Poggi de Aragão M., Reis M., Uchoa E. and Werneck R.F. (2006). Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem. *Mathematical Programming,* **106**, 3, 491–511.

[56] Gendreau M., Laporte G., Séguin, R. (1996). Stochastic Vehicle Routing. *European Journal of Operational Research,* **88**, 1, 3–12.

[57] Ghiani G., Laganà D., Manni E., Musmanno R. and Vigo D. (2011). Operations research in solid waste management: A survey. *Technical Report OR2011/10, D.E.I.S. - University of Bologna.*

[58] Gillet B.E. and Miller L.R. (1974). A heuristics algorithm for the vehicle dispatch problem. *Operation Research,* **22**, 340-349.

[59] Glover F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences,* **8**, 1, 156-166.

[60] Glover F., (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research,* **13**, 5, 533–549.

[61] Golden B.L., Raghavan S. and Wasil E., editors. (2007). The vehicle routing problem: latest advances and new challenges. *Springer.*

[62] Golden B.L. and Wong R.T. (1981). Capacitated arc routing problems. *Networks,* **11**, 3, 305-315.

[63] Golden B.L., Assad A.A., Levy L. and Gheysens F.G. (1984). The feet size and mix vehicle routing problem. *Computers & Operations Research,* **11**, 49–66.

[64] Golden B., Assad A. and Wasil E. (2001). Routing vehicles in the real world: Applications in the solid waste, beverage, food, dairy and newspaper industry. In Toth, P. and Vigo, D., editors, The Vehicle Routing Problem *SIAM, Philadelphia, PA,* 245-286.

[65] Golden B.L., Magnanti T.L. and Nguyen H.Q. (1977). Implementing vehicle routing algorithms. *Networks,* **7**, 113–148.

[66] Gouveia L., Mourão M.C. and Pinto L.S. (2010). Lower bounds for the mixed capacitated arc routing problem. *Computers & Operations Research,* **37**, 4, 692–699.

[67] Guan M. (1962). Graphic Programming Using Odd and Even Points. *Chinese Mathematics,* **1**, 273–277.

[68] Hansen M.P. and Jaszkiewicz A. (1998). Evaluating the quality of approximations of the nondominated set. *Technical report, Institute of Mathematical Modeling, Technical University of Denmark. IMM Technical Report IMM-REP-1998-7.*

[69] Hansen P. and Mladenović N. (1998). An introduction to variable neighbourhood search. *In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, (editors), Meta-heuristics, Advances and trends in local search paradigms for optimization,* 433-458. Kluwer Academic Publishers.

[70] Hansen P. and Mladenović N. (2001). Variable neighbourhood search: Principles and applications. *European Journal of Operational Research,* **130**, 3, 449-467.

[71] Hemmelmayr V.C., Doerner K.F., Hartl R.F. and Vigo D. (2012). Models and Algorithms for the Integrated Planning of Bin Allocation and Vehicle in Solid Waste Management. *Transportation Sciences,* , under revision..

[72] Hertz A. and Mittaz M. (2000) Heuristic Algorithms. *In: Arc Routing: Theory, Solutions and Applications,* 327-386.

[73] Hertz A. and Mittaz M. (2001) A variable neighborhood descent algorithm for the undirected capacitated arc routing problem. *Transportation Science,* **35**, 4, 425-434.

[74] Hierholzer C. (1873). Uber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahrem. *Mathematische Annalen,* **VI**, 30-31.

[75] Hirabayashi R., Nishida N. and Saruwatari Y. (1992). Tour construction algorithm for the capacitated arc routing problem. *Asia-Pacific Journal of Operational Research,* **9**, 155-175.

[76] Holland J.H. Adaptation in Natural and Artificial Systems. MIT Press, 1975.

[77] Jozefowiez N., Semet F. and Talbi E.-G. (2007). Target aiming Pareto search and its application to the vehicle routing problems with route balancing. *Journal of Heuristics,* **13**, 455-569.

[78] Jozefowiez N., Semet F. and Talbi E.-G. (2008). Multi-objective vehicle routing problems. *European Journal of Operational Research,* **189**, 2, 293-309.

[79] Kelly J.P. and Osman I.H. (1996). Meta-Heuristic: Theory and Applications. *Kluwer Academic Publisher.*

[80] Kirkpatrick S., Gelatt C.D. and Vecchi M.P. (1983). Optimization by simulated annealing. *Science,* **220**, 671-680.

[81] Knowles J., Thiele L. and Zitzler E. (2006). A tutorial on the performance assessment of stochastic multiobjective optimizers. Technical report, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland.

[82] Kruskal W.H., Wallis W.A. (1952).Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association,* **47**, 260, 583–621.

[83] Kytöjoki J., Nuortio T., Bräysy O. and Gendreau M. (2007). An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research,* **34**, 2743-2757.

[84] Knight K. and Hofer J. (1968). Vehicle scheduling with timed and connected calls: A case study. *Operational Research Quarterly,* **19**, 299-310.

[85] Lacomme P., Prins C. and Sevaux M. (2006). A genetic algorithm for a bi-objective capacitated arc routing problem. *Computers & Operations Research,* **33**, 3473–3493.

[86] Laporte G. (1986). Generalized Subtour Elimination Constraints and Connectivity Constraints. *Journal of the Operational Research Society,* **37**, 5, 509–514.

[87] Laporte G., Norbert Y. and Desrochers M. (1984), Two Exact Algorithms for the Distance Constrained Vehicle Routing Problem. *Networks,* **14**, 47–61.

[88] Laporte G., Norbert Y. and Desrochers M. (1985). Optimal routing under capacity and distance restrictions. *Operations Research,* **33**, 1058–1073.

[89] Laporte G., Mercure H. and Norbert Y. (1992). A branch-and-bound algorithm for a class of asymmetrical vehicle routing problems. *Journal of the Operational Research Society,* **43**, 5, 469–481.

[90] Letchford A.N. and Oukil A. (2009). Exploiting sparsity in princing routines for the capacitated arc routing problems. *Computers & Operations Research,* **36**, 7, 2320–2327.

[91] Li J.-Q., Borenstein D. and Mirchandani P.B. (2008). Truck scheduling for solid waste collection in the City of Porto Alegre, Brazil. *Omega,* **36**, 1133–1149.

[92] Lin S.W., Yu V.F. and Lu C.C. (2011). A simulated annealing heuristic for the truck and trailer routing problem with time windows. *Expert Systems with Applications,* **38**, 15244-15252.

[93] Longo H., Poggi de Aragão M., and Uchoa E. (2006). Solving capacitated arc routing problems using a transformation to the CVRP. *Computers & Operations Research,* **33**, 6, 1823–1837.

[94] Lysgaard J., Letchford A. and Eglese R. (2004). A new branch-and-cut algorithm for capacitated vehicle routing problems. *Mathematical Programming,* **100**, 2, 423–445.

[95] Maniezzo V. and Roffilli M. (2008). Algorithms for Large Directed Capacitated Arc Routing Problem Instances. *Cotta C. and van Hemert J. (Editors): Recent Advances in Evolutionary Computation for Combinatorial Optimization* **153**, 259–274.

[96] Maniezzo V., Stützle T. and Voβ S., editors. (2010). Matheuristics: Hybridizing Metaheuristics and Mathematical Programming. *Annals of Information Systems, Springer,* **10**.

[97] Martí R., Campos V., Resende M.G.D. and Duarte A. (2011). Multi-Objective GRASP with Path Relinking. *Manuscript sumbitted for publication.*

[98] Melián B., Moreno-Pérez J.A. and Moreno-Vega M. (2003). Metaheurísticas: una visión global. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial,* **7**, 19, 7–28.

[99] Min H. (1989). The multiple vehicle routing problem with simultaneous delivery and pick-up points. *Transportation Research,* **23**, 5, 377–386.

[100] Miler D.L., Tucker A.W. and Zemlin R.A. (1960). Integer programming formulations and traveling salesman problems. *Journal of the ACM,* **7**, 326–329.

[101] Minieka E. (1979). The Chinese Postman Problem for Mixed Networks. *Management Science,* **25**, 643–648.

[102] Mole R.H. and Jameson S.R. (1976). A sequential route-building algorithm employing a generalized saving criterion. *Operation Research Quarterly,* **27**, 503–511.

[103] Mourão M.C. and Almeida M.T. (2000). Lower-bounding and heuristic methods for a refuse collection vehicle routing problem. *European Journal of Operational Research,* **121**, 420–434.

[104] Mourão M.C. and Amado L. (2005). Heuristic method for a mixed capacitated arc routing problem: A refuse collection application. *European Journal of Operational Research,* **160**, 139–153.

[105] Naddef D. and Rinaldi G. (2001). Branch-and-cut algorithms for the capacitated vrp. *In P. Toth and D. Vigo, editors, The vehicle routing problem, SIAM, Philadelphia, PA. Chapter 3,* 53–84.

[106] Ombuki B., Ross B.J. and Hanshar F. (2006). Multi-objective genetic algorithm for vehicle routing problem with time windows. *Applied Intelligence,* **24**, 1, 17-30.

[107] Orloff C.S. (1974). A fundamental problem in vehicle routing. *Networks,* **4**, 1, 35-64.

[108] Pacheco J., Martí R. (2006). Tabu Search for a Multi-Objective Routing Problem. *Journal of the Operational Research Society,* **57**, 29-37.

[109] Park Y. and Koelling C. (1986). A solution of vehicle routing problems in multiple objective environment. *Engineering Costs and Production Economics,* **10**, 121–132.

[110] Park Y. and Koelling C. (1989). An interactive computerized algorithm for multicriteria vehicle routing problems. *Computers & Industrial Engineering,* **16**, 477–490.

[111] Pearn W.L., Assad A.A. and Golden B.L. (1987). Transforming Arc Routing into Node Routing Problems. *Computers & Operations Research,* **14**, 4, 285-288.

[112] Pisinger D. and Ropke S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research,* **34**, 8, 2403–2435.

[113] Potters, J.A.M., Curiel, I.J. and Tijs, S.H. (1992). Traveling salesman games. *Mathematical Programming,* **53**, 199-211.

[114] Psaraftis H.N. (1995) Dynamic vehicle routing: Status and prospects. *Annals of Operations Research,* **61**, 143–164.

[115] Pullen H. and Webb M. (1967) A Computer Application to a Transport Scheduling Problem. *Journal of Computing,* **10**, 10–13.

[116] Rahoual M., Kitoun B., Mabed M., Bachelet V. and Benameur F. (2001) Multicriteria genetic algorithms for the vehicle routing problem with time windows. *In: Metaheuristic International Conference (MIC2001),* 527-532.

[117] Raidl G. (2006). A unified view on hybrid metaheuristics, *In: Proceedings of HM 2006, Springer LNCS,* **4030**, 1-12.

[118] Reiter P. and Gutjaht W.J. (2012). Exact hybrid algorithms for solving a bi-objective vehicle routing problem. *Central European Journal of Operations Research,* **20**, 19-43.

[119] Ribeiro R. and Lourenço H.R. (2001). A multi-objective model for a multi-period distribution management problem. *In Metaheuristic International Conference 2001,* 91-102.

[120] Salani M. (2006). Branch-and-Price Algorithms for Vehicle Routing Problems. *Università degli studi di Milano.*

[121] Santos L., Coutinho-Rodrigues J. and Current J.R. (2010). An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Transportation Research,* **44**, 246–266.

[122] Schrage L. (1981). Formulation and structure of more complex/realistic routing and scheduling problems. *Networks,* **11**, 229–232.

[123] Semet F. and Laporte G. (2002). Classical heuristics for the capacitated vrp. The Vehicle Routing Problem. *Monographs on Discrete Mathematics and Aplications. SIAM, Philadelphia, PA.* **9**, 109-128.

[124] Sessomboon W., Watanabe K., Irohara T. and Yoshimoto K. (1998). A study on multi-objective vehicle routing problem considering customer satisfaction with due-time (the creation of Pareto optimal solution by hybrid genetic algorithm). *Transaction of the Japan Society of Mechanical Engineering.*

[125] Simonetto E.O. and Borenstein D. (2007). A decision support system for the operational planning of solid waste collection. *Waste Management,* **27**, 10, 1286–1297.

[126] Taillard É.D. (1999). A heuristic column generation method for heterogeneous fleet. *RAIRO (Recherche Opérationnelle),* **33**, 1–14.

[127] Tillman F.A. (1969). The multiple terminal delivery problem with probabilistic demands. *Transportation Science,* **3**, 192–204.

[128] Toth P. and Vigo D., editors. (2002). The Vehicle Routing Problem. *Monographs on Discrete Mathematics and Aplications. SIAM, Philadelphia, PA.*

[129] Villegas J.G., Prins C., Prodhon C., Medaglia A.L. and Velasco N. (2011). A GRASP with evolutionary path relinking for the truck and trailer routing problem. *Computers & Operations Research,* **38**, 1319-1334.

[130] Vigo D., Bonoli A., Gricinella A. and Zarri G. (2007). Key Elements for Optimal Integrated Urban Solid Waste Management. *International Experience. Esculapio, Bologna.*

[131] Winston W. L. (1994). Operations research: applications and algorithms. *Duxbury Press, Belmont, CA, 3rd edition.*

[132] Zitzler E. (1999). Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. PhD thesis, Swiss Federal Institute of Technology (ETH).

[133] Zitzler E. and Thiele L. (1998). Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. *In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, Springer, Heidelberg* **1498**, 292–301.

[134] Zitzler E., Thiele L., Laumanns M., Foneseca C.M. and Grunert da Fonseca V. (2003). Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation,* **7**, 2, 117-132.